普通高等教育"计算机类专业"规划教材

软件测试基础
与测试案例分析

张　坤　李　媚　王　向　主　编
阮冬茹　高　凯　高国江　副主编

清华大学出版社

普通高等教育"计算机类专业"规划教材

软件测试基础与测试案例分析

张　坤　李　媚　主编

阮冬茹　高　凯　王　向　高国江　副主编

清 华 大 学 出 版 社

北　京

内 容 简 介

本书从多个视角对软件测试技术与方法进行阐述，内容涵盖软件测试基础、测试用例设计、集成测试、系统测试、测试文档写作、黑盒测试与白盒测试、UML 建模、有限状态机、Petri 网和状态图等。全书内容综合全面，理论性强，体系完整，内容新颖，条理清晰，组织合理，强调实践。本书可作为高校相关专业（如计算机科学与技术、软件工程、信息管理与信息系统）相关课程的教材，同时对于从事计算机软件开发的工程技术人员和希望了解软件测试技术的爱好者也具有较高的参考价值。

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。
版权所有，侵权必究。侵权举报电话：010-62782989　13701121933

图书在版编目（CIP）数据

软件测试基础与测试案例分析/张坤，李媚，王向主编.--北京：清华大学出版社，2014
普通高等教育"计算机类专业"规划教材
ISBN 978-7-302-35876-3

Ⅰ. ①软…　Ⅱ. ①张… ②李… ③王…　Ⅲ. ①软件－测试－高等学校－教材　Ⅳ. ①TP311.5

中国版本图书馆 CIP 数据核字（2014）第 060990 号

责任编辑：白立军　战晓雷
封面设计：常雪影
责任校对：时翠兰
责任印制：王静怡

出版发行：清华大学出版社
　　网　　址：http://www.tup.com.cn，http://www.wqbook.com
　　地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
　　社 总 机：010-62770175　　**邮　　购：**010-62786544
　　投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
　　质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
　　课件下载：http://www.tup.com.cn，010-62795954
印 装 者：三河市李旗庄少明印装厂
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张：**16　　**字　　数：**370 千字
版　　次：2014 年 9 月第 1 版　　**印　　次：**2014 年 9 月第1 次印刷
印　　数：1～2000
定　　价：29.50 元

产品编号：050330-01

前 言

随着计算机系统规模和复杂性的急剧增加，计算机软硬件出现故障和系统失效的可能性也在增加。为保证计算机软件的质量，软件测试正日益受到 IT 业的重视。软件测试一般是指在规定的条件下，对计算机软件进行测试，发现其中可能存在的错误，并对其是否能满足设计要求进行评估的过程。针对现今复杂度高、规模大的计算机软件产品，如何进行高效的专业化测试，已成为业内人士所关心的问题之一。对于从事计算机教学、科研、工程开发、软件产品应用等领域的人来说，掌握常见软件测试工具的使用方法，非常有必要。

本书理论性强，体系完整，内容新颖，条理清晰，组织合理，实践性强，从多个视角对计算机软件测试技术进行了分析。内容涵盖与计算机软件测试相关的多个重要部分，包括软件测试流程、软件测试用例设计和管理工具使用等。全书共分 10 章。第 1 章介绍了软件测试基础知识及 4 种不同的测试模型与相应测试过程中的步骤，分析了软件测试现状，提出了软件测试的职业发展方向；第 2 章叙述了书中所用到的测试用例；第 3 章介绍了测试用例的设计方法并分析了针对不同用例的设计方法；第 4 章介绍了集成测试方法，介绍了 MM 路径集成测试的实际应用过程；第 5 章对系统测试的各个方面进行了说明，分析了性能测试、压力测试、容量测试以及 GUI 测试等；第 6 章介绍了软件测试的流程以及各种测试文档的写作要求；第7 章介绍了黑盒测试工具，并介绍了 IBM Rational Function Tester 工具的使用；第 8 章介绍了软件测试中的白盒测试法，并以 JUnit 和 HtmlUnit 为例，介绍了基本的测试框架和一些高级应用，最后以一个完整的 NextDate 问题为例，演示了 JUnit 测试方法的实际应用；第 9 章介绍了软件性能测试前的准备、性能测试工具等；第 10 章介绍了 IBM Rational ClearQuest 工具的使用，可用来实现对软件的缺陷跟踪管理。除上述内容外，本书内容还涉及 UML 建模、面向对象软件测试、有限状态机、Petri 网和状态图等的应用。全书内容综合全面，侧重工程实践，结合有针对性的案例，可帮助读者了解软件测试的理论与实践过程。

作者团队以认真、严谨的科学态度实现了书中绝大部分的主要方法，尽量详尽地描述了各种方法的适用环境以及取得的效果。很多老师、同事、学生也花费了大量的时间帮助我们审阅和组织了与其研究领域相关的章节内容。本书的写作分工如下：张坤撰写了第 5 章，第 7 章和第 10 章的主要内容；李媚撰写了第 4 章，第 8 章和第 9 章中的主要内容；张坤和李媚合作完成了第 2 章和第 3 章的主要内容及相关案例；王向撰写了第 1 章和第 6 章。最后，阮冬茹和高凯审阅了全书并提出修改意见。在本书的写作与相关科研课题的研究工作中，得到了多方面的支持与帮助。这里要特别感谢杨奎河、高国江、马红霞等老师提供的帮助，感谢作者团队指导的研究生同学的辛勤付出，也感谢众多研究生忍受我们的各种严格要求。

本书的顺利完成也得益于作者参阅了大量的相关工作及研究成果，在此谨向这些文献的作者以及为本书提供帮助的老师、同仁、学生和课题组成员致以诚挚的谢意。在本书写作过程中，也得到了清华大学出版社白立军等的大力支持和帮助，在此一并表示衷心感谢。

由于我们的学识、水平所限，书中不妥之处在所难免，恳请广大读者批评指正。

编著者

2014 年 5 月

目录

第 1 章　软件测试概述

在计算机技术飞速发展的今天，人们对计算机的需求和依赖与日俱增。随着软件规模的越来越大，越来越复杂，软件的可靠性越来越难以保证，软件的生产成本以及软件中存在的缺陷和故障造成的各类损失大大增加，甚至会带来灾难性的后果。因此，软件可靠性问题成为所有使用软件和开发软件的人关注的焦点问题，在展望 21 世纪计算机科学发展方向和策略的同时，许多科学家把软件质量放在优先于提高软件功能和性能的位置上。

衡量软件质量的有效方法就是软件测试。软件测试是软件开发过程中的重要步骤，是对需求分析、设计规格说明和编码的最终审定，对保证软件的质量起到关键的作用。随着软件系统规模不断加大，复杂性不断提高，进行专业化高效软件测试的要求越来越严格，软件测试职业的价值逐步得到了认可。新的测试理论、测试方法和测试技术手段在不断地涌现，软件测试机构和组织以及测试从业人员群体也在迅速产生和发展，软件测试已经作为一门新兴技术完善和健全起来。

1.1　计算机软件可靠性问题

在当今社会，计算机技术迅速发展，越来越广泛地应用于国民经济和社会生活的各个方面，随着软件系统规模和复杂性与日俱增，软件的可靠性已经越来越受到重视，成为现今社会关注的一个重要问题。应用本身对系统运行的可靠性要求越来越高，在一些关键的应用领域，如航空、航天等，软件的可靠性尤为重要。在一些敏感服务性行业，比如银行等，软件系统的可靠性更是直接关系到其声誉和生存发展竞争能力。软件可靠性比硬件可靠性更难保证，软件设计故障引起的系统失效大约是计算机硬件设计故障引发的系统失效的 10 倍，会严重影响整个系统的可靠性。在许多项目开发过程中，对可靠性没有提出明确的要求，开发商或者部门不会在可靠性上投入更多的精力，往往只注重软件开发的速度、结果的正确性和用户界面的友好性等，而忽略了可靠性。在软件投入使用后才发现大量可靠性问题，增加了维护困难和工作量，当问题严重时只有将软件束之高阁，无法投入实际使用。只有软件的可靠性大幅度提高了，才能使计算机广泛应用于社会的各个方面。

一个可靠的软件应该是正确、完整、一致和健壮的，这也是用户所期望的。IEEE 将软件可靠性定义为：系统在特定环境下，在给定的时间内无故障运行的概率。因此，软件可靠性是对软件在设计、开发以及所预定的环境下具有能力的置信度的一个度量，它是衡量软件质量的主要参数之一。所以软件测试是保证软件质量，提高软件可靠性的最重要手段。目前，软件测试在整个软件开发周期中所占的比例日益上升，许多软件开发组织已经将开发资源的 40%用于软件测试中。对于高可靠性的软件，如飞行控制、军事武器系统、核反应堆控制和金融软件等，其软件测试费用超过软件开发其他阶段所用费用的总和，甚至达到后者的 3～5 倍。

软件是人脑高度智力化的体现，和其他科技和生产领域不同的是，软件与生俱来就有可

能存在某种缺陷,虽然软件开发者会采取各种各样的有效措施来提高软件开发的质量,尽量避免缺陷,但是软件的缺陷和故障还是存在的。下面通过几个实例来说明软件缺陷和故障问题可能造成的严重损失和灾难。

1. 飞行事故

众所周知,飞机的安全飞行是由飞行控制软件直接控制的,因此,一旦飞行控制软件发生错误,则后果不堪设想。1994 年在苏格兰,一架吉努克型直升飞机坠毁,29 名乘客全部罹难。最初指责声都指向飞行员,但后来有证据表明,直升飞机的系统错误才是罪魁祸首。

另外一次因软件而引发的飞行事故发生在 1993 年,瑞典的一架 JAS 39 鹰狮战斗机因飞行控制软件的错误(bug)而坠毁。

2. 一触即发的第三次世界大战

1980 年,北美防空联合司令部曾报告称美国遭受导弹袭击。后来证实,这是反馈系统的电路发生故障而引起的误报。

1983 年,苏联卫星报告有美国导弹入侵,但主管官员的直觉告诉他这是误报。后来事实证明的确是误报,同样也是由于软件故障引起的。

在上述两个案例中,幸亏这些误报没有激活"核按钮",否则核战争将全面爆发,后果将不堪设想。

3. "漏网"的臭氧层空洞

1978 年,NASA(美国国家航空和宇宙航行局)启动了臭氧层测绘计划。但是直到 1985 年英国科学家在南极洲上方发现了很大的臭氧层空洞,NASA 还一直没有发现这一问题。对于这个很大臭氧层空洞的"漏网",主要原因在于:在设计的时候,用于该计划的数据分析软件忽略了那些和预测值有很大差距的数据,直到 NASA 重新检测它们的数据时才发现这一错误。修正错误后,NASA 才证实南极臭氧层的确有一个很大的空洞。

4. 致命的辐射治疗

1985—1987 年,Therac-25 辐射治疗设备卷入多宗因辐射剂量严重超标引发的医疗事故,其罪魁祸首是医疗设备电力软件的错误。据统计,大量患者接受了高达 100 倍的预定剂量的辐射治疗,其中至少 3 人直接死于辐射剂量超标。

另一宗辐射剂量超标的事故发生在 2000 年的巴拿马城(巴拿马首都)。从美国 Multidata 公司引入的治疗规划软件,其辐射剂量的预设值有误。有些患者接受了超标剂量的治疗,至少有 5 人死亡。后续几年中,又有 21 人死亡,但很难确定这 21 人中到底有多少人是死于本身的癌症,还是辐射治疗剂量超标引发的不良后果。

5. 阿丽亚娜 5 型火箭的悲剧处女秀

1996 年 6 月 4 日,阿丽亚娜 5 型运载火箭首航,原计划运送 4 颗太阳风观察卫星到预定轨道,但由于软件引发的问题导致火箭在发射 39 秒后偏轨,从而激活了火箭的自我摧毁装置,阿丽亚娜 5 型火箭和其他卫星在瞬间灰飞烟灭,如图 1.1 所示。


图 1.1 阿丽亚娜 5 型火箭爆炸瞬间

后来查明的事故原因是代码重用。阿丽亚娜 5 型火箭的发射系统代码直接重用了阿丽亚娜 4 型火箭的相应代码,而阿丽亚娜 4 型火箭的飞行条件和阿丽亚

娜5型火箭的飞行条件截然不同。此次事故损失3.7亿美元。

6. 错误的高考成绩

在国内也有不少由于软件缺陷而造成的严重后果。2007年,安徽省高考文科综合科目成绩因计算机软件的失误,漏统了第39题考分,致使文科考生成绩统计有误。这不仅给考生和家长带来了巨大的心理压力,而且为考试院后期的补救工作带来了沉重的负担。

全球每年因软件缺陷引发的问题数不胜数,以上的几个例子只是冰山一角。巨额的财产损失和生命的代价让人们越来越重视软件的质量。当前,软件产品应用到了社会的各个领域,软件开发商为了占领市场,必须提高软件质量,以免在激烈的竞争中被淘汰。用户为了保证自己业务的顺利完成,当然希望选用优质的软件。质量欠佳的软件产品不仅会使开发商的维护费用和用户的使用成本大幅增加,还可能产生其他的责任风险,造成公司信誉下降。因此,为了防止和减少软件缺陷,提高软件质量,必须进行软件测试。软件测试是最有效的排除和防止软件缺陷和故障的手段,并且软件测试实践促进了软件测试理论的快速发展以及软件测试技术的不断完善和健全。

1.2 软件测试的基本知识

1.2.1 软件测试背景

软件测试是伴随着软件的产生而产生的。在计算机行业发展初期,软件测试就已经开始了。但是早期的软件开发过程中,软件规模较小,复杂程度低,软件开发过程相当随意,开发人员进行的只是类似调试的测试,目的是纠正软件中已知的故障,由开发人员完成这方面的工作。测试没有任何计划和方法,测试用例由开发人员根据经验随机设计和选取,一般是在产品已经基本完成时才进行测试。

一直到1957年,软件测试和调试才分离开来,测试仍然是在软件生命周期的最后进行。但缺乏有效的测试方法,主要依靠“错误推测”来寻找软件中的缺陷。这一时期软件测试的理论和方法发展得比较缓慢。

20世纪70年代,随着软件开发技术的成熟和完善,软件的规模不断加大,复杂性也大为提高,软件的可靠性面临着前所未有的危机,给软件测试工作带来了更大的挑战,很多测试理论和测试方法应运而生,逐渐形成了一套完整的体系。1972年,Bill Hetzel在《软件测试完全指南》(*Complete Guide of Software Testing*)一书中指出:“测试是以评价一个程序或者系统属性为目标的任何一种活动。测试是对软件质量的度量。”这个定义至今仍然被引用。1979年,Glenford J. Myers认为,测试应该首先认定软件是有错误的,然后用逆向思维去发现尽可能多的错误。他在《软件测试的艺术》(*The Art of Software Testing*)一书中提出了对软件测试的定义:“测试是为了发现错误而执行一个程序或者系统的过程。”

如今,人们对软件质量、成本和进度的要求越来越高,质量控制已经不仅仅是传统意义上的软件测试。随着软件趋向大型化、高复杂度,软件的质量越来越重要。人们开始为软件开发设计了各种流程和管理方法,软件开发的方式逐渐过渡到结构化的开发过程,以结构化分析与设计、结构化评审、结构化程序设计以及结构化测试为特征。在整个软件开发过程中,测试已经不再只是基于程序代码进行的活动,而是一个基于整个软件生命周期的质量控

制活动，贯穿于软件开发的各个阶段，成为软件质量保证(SQA)的主要职能。

软件测试的重要性越来越被人们接受，甚至出现了软件开发活动应该以测试为主导的思潮。软件测试的重要性主要体现在：寻找软件错误，以便进行修正；验证软件是否符合要求；指导软件的开发过程；提供软件的相关特征。因此，不难看出，软件测试在整个软件开发过程中是不可或缺的。

1.2.2 软件测试的原则

从用户的角度出发，希望通过软件测试充分暴露软件中存在的问题和缺陷；从开发者的角度出发，希望测试能表明软件产品不存在错误，已经正确地实现了用户的需求。针对软件测试中重要的问题，可以归纳出以下的测试指导原则，这些原则看上去大多都是显而易见的，但是常常被忽略。

1. 应当把“尽早和不断的测试”作为开发者的座右铭

软件测试是贯穿于整个软件开发过程的，应该在测试工作真正开始前的较长时间内就执行测试计划。测试计划可以在需求模型一完成就开始，详细的测试用例定义可以在设计模型被确定后立即开始。因此，所有测试应该在产生任何代码前就开始计划和设计。

2. 程序员应该避免检查自己的程序

程序设计机构不应测试自己的程序，程序员也应避免测试自己的程序。软件测试的出发点是寻找错误，当程序员“建设性”地设计和编写完程序之后，很难让他突然转变视角，以一种“破坏性”的眼光来审查程序。另外，如果程序员在软件开发过程中错误理解了定义或者规范，导致程序中出现错误，他在测试的过程中会带着同样的误解来测试自己的程序。因此，测试工作应该由独立的、专业的软件测试机构来完成，这样会更加有效，更容易测试成功。

3. 完全测试是不可能的，需要终止

任何进行测试的人，都希望对软件进行完全测试，找出所有软件缺陷，使软件臻于完美。但是测试并不能显示软件潜在的缺陷，“测试只能证明软件存在错误而不能证明软件没有错误。”最初的测试通常把焦点放在单个程序模块上，进一步测试的焦点则转向在集成的模块簇中寻找错误，最后在整个系统中寻找错误。在测试中运行路径的每一种组合是不现实的。

4. 确定预期输出或结果是测试情况必不可少的一部分

如果事先无法肯定预期的测试结果，往往会把看起来似是而非的不确定的输出当成正确的结果。因此，必须提倡用事先精确对应的输入和输出结果来详细检查所有的输出。

5. 应当彻底检查每个测试的执行结果

在最终发现的错误中，往往会有一大部分在前面的测试中已经暴露了出来，但是在软件测试的时候却没有把这些错误找出来，这些错误正是前面的测试所遗漏的。因此，对于每一个测试的执行结果必须彻底检查。

6. 设计测试用例时针对有效、预期的输入和无效、未预料的输入两种情况

在测试软件时，人们往往倾向于将重点集中在有效和预期的输入情况上，而忽略了无效和未预料到的情况。例如，在第 2 章的 NextDate 程序的测试中，总是出现这个倾向。很少有人设计测试用例输入无效的年月日，来得到无效的结果。在软件产品中突然暴露出来的许多问题是当程序以某些新的或未预料到的方式运行时发现的。因此，针对无效或者未预

料到的情况的测试用例，比针对有效输入情况的测试用例更能够发现问题。

7. 检查程序不仅要看它是否做了该做的事，还要看它是否做了不该做的事

在测试程序时，必须检查程序是否有我们不希望看到的副作用。例如在ATM(自动取款机)系统中，虽然生成了正确的取款单，但是是为非法用户生成的或者覆盖了其他用户的记录，这样的程序都是不正确的程序。

8. 避免测试用例用后即弃

对程序做了修改以后，为了提高对程序重新测试的效率，减少测试工作量和成本，测试用例一般不要丢掉。测试用例代表了一定的价值投资。保留测试用例，当程序某些部件发生更改以后重新执行，这就是"回归测试"。

9. 一段程序存在错误的概率和在此已发现的错误数成正比

错误总是倾向于聚集存在，而在一个具体的程序中，某些部分要比其他部分更容易存在错误。因此，为了使测试获得更大的成效，最好对那些容易存在错误的部分进行额外的测试。

1.2.3 软件测试的分类

软件测试的分类方法很多，从不同的角度可以对软件测试进行不同的分类。这里简单介绍常用的分类。

1. 根据开发过程分类

从软件的开发过程来分，软件测试可以分为以下几类。

(1) 单元测试(unit testing)：是指对软件中的最小可测试单元进行检查和验证。对于单元测试中的单元，一般来说，要根据实际情况去判定其具体含义。如C语言中单元指一个函数，Java中单元指一个类，图形化的软件中可以指一个窗口或一个菜单等。总的来说，单元就是人为规定的最小的被测功能模块。单元测试是在软件开发过程中要进行的最低级别的测试活动，软件的独立单元将在与程序的其他部分相隔离的情况下进行测试。

单元测试一般由编程人员自行编制测试计划并执行测试，有时由白盒测试工程师负责。单元测试通常在编程阶段进行，必要时需要编写测试驱动器。

对于程序员来说，如果养成了对自己写的代码进行单元测试的习惯，不但可以写出高质量的代码，而且能提高编程水平。

(2) 集成测试(integration testing)：也叫组装测试或联合测试。在单元测试的基础上，将所有模块按照设计要求(如根据结构图)组装成为子系统或系统，进行集成测试。实践表明，一些模块虽然能够单独地工作，但并不能保证连接起来也能正常工作。程序在某些局部反映不出来的问题，在全局上很可能暴露出来，影响功能的实现。

软件单元按照设计功能划分为模块，集成测试是对模块的功能、性能及模块与模块间的接口进行测试。集合测试的组织，即选择什么方式把模块组装起来形成一个可运行的系统，直接影响到模块测试用例的形式、所用测试工具的类型、模块编号和测试的次序、生成测试用例和调试的费用等。常用的组装方式是一次性组装方式和增量式组装方式。

集成测试应由专门的测试小组进行，测试小组由有经验的系统设计人员和程序员组成。整个测试活动要在评审人员出席的情况下进行。

(3) 系统测试(system testing)：是将已经确认的软件、计算机硬件、外设和网络等其他

元素结合在一起,进行信息系统的各种组装测试和确认测试。系统测试是针对整个产品系统进行的测试,目的是验证系统是否满足需求规格的定义,找出与需求规格不符或与之矛盾的地方,从而提出更加完善的方案。系统测试发现问题之后要经过调试找出错误原因和位置,然后进行改正。系统测试属于黑盒测试的范畴。

系统测试主要由测试部门完成。

(4) 用户验收测试(user acceptance testing):是用户在测试人员的协助下根据测试计划和结果对系统进行测试和接收。主要是验证软件的功能、性能及特性是否与用户的要求一致。以软件功能需求说明书及用户手册为标准来测试整个系统,保证软件达到可以交付使用的状态。

(5) 回归测试(regression testing):是指修改了旧代码后,重新进行测试以确认修改没有引入新的错误或导致其他代码产生错误。自动回归测试将大幅降低系统测试、维护升级等阶段的成本。回归测试作为软件生命周期的一个组成部分,在整个软件测试过程中占有很大的工作量比重,软件开发的各个阶段都会进行多次回归测试。在渐进和快速迭代开发中,新版本的连续发布使回归测试进行得更加频繁;而在极端编程方法中,更是要求每天都进行若干次回归测试。因此,通过选择正确的回归测试策略来改进回归测试的效率和有效性是非常有意义的。

回归测试也是由专门的测试人员来完成的。

2. 根据测试特性分类

从测试特性上对软件测试进行分类,主要分为白盒测试、黑盒测试和灰盒测试。

可以形象地把软件测试比喻为对一个盒子的检测。盒子里面装着东西,对盒子的检测可以有3种情况:如果盒子是一个密封的黑盒子,那么检测这个盒子只能通过它的外形,根据它摇晃的声音来判断里面内容的好坏,但是往往检测得不准确。而使用白盒测试,则把盒子看成一个玻璃盒子,不仅能够清楚地看到里面的一切,而且如果发现里面的东西不对时,还可以把盒子里面的东西重新放好。而灰盒子则是介于两者之间。下面对软件测试的三大类型的特性进行具体讲解。

(1) 白盒测试(white-box testing):通过程序的源代码进行测试。这种类型的测试需要从代码句法发现内部代码在算法、溢出、路径、条件等问题中的缺点或错误,进而加以修正。因此,白盒测试要求测试工程师对软件的内部结构及逻辑有深入的了解,并掌握编写该源程序的语言。这类测试包括语句测试、分支测试、路径测试、条件测试和目测等方法。

(2) 黑盒测试(black-box testing):在测试时,把程序看作一个不能打开的黑盒子,在完全不考虑程序内部结构和内部特性的情况下,测试者在程序接口进行测试,它只检查程序功能是否能够按照需求规格说明书的规定正常使用,程序是否能适当地接收和正确地输出。这类测试主要是功能测试。

(3) 灰盒测试(grey-box testing):介于白盒测试和黑盒测试之间,灰盒测试关注输出对于输入的正确性,也关注内部表现,但这种关注不像白盒那样详细、完整,只是通过一些表征性的现象、事件和标志来判断内部的运行状态。有时候输出是正确的,但内部其实已经有错误了,这种情况非常多。如果每次都通过白盒测试来操作,效率会很低,因此需要采取这样的灰盒测试。

3. 根据是否运行程序分类

根据是否运行程序对软件测试进行分类,可以分为静态测试和动态测试两类。

(1) 静态测试(static testing):是指不运行被测程序,仅通过分析或检查源程序的语法、结构、过程和接口等来检查程序的正确性。对需求规格说明书、软件设计说明书和源程序做结构分析、流程图分析和符号执行来找错。静态方法通过对程序静态特性的分析,找出欠缺和可疑之处,例如不匹配的参数、不适当的循环嵌套和分支嵌套、不允许的递归、未使用过的变量、空指针的引用和可疑的计算等。静态测试结果可用于进一步的查错,并为测试用例选取提供指导。

(2) 动态测试(dynamic testing):就是通过运行软件来检验软件的动态行为和运行结果的正确性。动态测试是测试工作的主要方式。

4. 根据功能测试分类

功能测试就是对产品的各项功能进行验证,根据功能测试用例,逐项测试,检查产品是否达到用户要求的功能。功能测试有多种方法,主要分为以下几类。

(1) 界面测试(UI testing):界面是软件与用户交互的最直接的层,界面的好坏决定用户对软件的第一印象。设计良好的界面能够引导用户完成相应的操作,起到向导的作用。因此对于界面的测试是十分必要的。

(2) 业务逻辑测试(business logic testing):不同的项目有不同的功能,不同的功能需要不同的实现,实现这些核心功能的代码就叫做业务逻辑。而测试这些功能有没有实现,就叫做业务逻辑测试。

(3) 兼容测试(compatibility testing):兼容测试是指测试软件在特定的硬件平台上、不同的应用软件之间、不同的操纵系统平台上以及不同的网络等环境中是否能够很友好地运行。

(4) 易用性测试(usability testing):是指用户使用软件时是否感觉方便。可以采用静态测试,也可以采用动态测试,或者动静结合的方式进行测试。

(5) 安全测试(security testing):是在软件产品的生命周期中,特别是产品开发基本完成,进入发布阶段,对产品进行检验以验证产品是否符合安全需求定义和产品质量标准的过程。

(6) 安装测试(installation testing):确保该软件在正常情况和异常情况的不同条件下都能进行安装。异常情况包括磁盘空间不足、缺少目录创建权限等。核实软件在安装后可立即正常运行。安装测试包括测试安装代码以及安装手册。安装手册提供安装指导,安装代码提供程序能够运行的基础数据。

5. 根据性能测试分类

根据性能对软件测试进行分类,可以分为以下几类。

(1) 负载测试(load testing):通过测试系统在资源超负荷情况下的表现,以发现设计上的错误或验证系统的负载能力。在这种测试中,将使测试对象承担不同的工作量,以评测和评估测试对象在不同工作量条件下的性能行为,以及持续正常运行的能力。负载测试的目标是确定并确保系统在超出最大预期工作量的情况下仍能正常运行。此外,负载测试还要评估性能特征,例如响应时间、事务处理速率和其他与时间相关的方面。

(2) 压力测试(pressure testing):是确立系统稳定性的一种测试方法,通常在系统正常

运作范围之外进行,以考察其功能极限和隐患。

(3) 容量测试(capacity testing):通过测试预先分析出反映软件系统应用特征的某项指标的极限值(如最大并发用户数、数据库记录数等),要求系统在其极限状态下不出现任何软件故障或还能保持主要功能正常运行。容量测试还将确定测试对象在给定时间内能够持续处理的最大负载或工作量。软件容量的测试能让软件开发商或用户了解该软件系统的承载能力或提供服务的能力,如某个电子商务网站所能承受的同时进行交易或结算的在线用户数。如果系统的实际容量不能满足设计要求,就应该寻求新的技术解决方案,以提高系统的容量。

(4) 并发测试(concurrent testing):主要指当测试多用户并发访问同一个应用、模块或数据时是否产生隐藏的并发问题,如内存泄漏、线程锁和资源争用等问题,几乎所有的性能测试都会涉及并发测试。

(5) 配置测试(configuration testing):主要是针对硬件而言,其测试过程是测试目标软件在具体硬件配置情况下是否出现问题。

(6) 可靠性测试(reliability testing):也称软件的可靠性评估,指根据软件系统可靠性结构(单元与系统间可靠性关系)、寿命类型和各单元的可靠性试验信息,利用概率统计方法,评估出系统的可靠性特征量。

软件可靠性是软件系统在规定的时间内以及规定的环境条件下完成规定功能的能力。一般情况下,只能通过对软件系统进行测试来度量其可靠性。

除了以上这些分类,还有一些其他的测试,比如冒烟测试、随机测试等。冒烟测试是在将代码更改嵌入到产品的源代码之前对这些更改进行验证的过程。在检查了代码后,冒烟测试是确定和修复软件缺陷的最经济有效的方法。随机测试主要是根据测试者的经验对软件进行功能和性能抽查。在软件开发的生命周期中,上述所有的测试都有可能用到,无论是哪一种测试,都是为了发现尽可能多的错误。测试人员的职责就是选择合适的测试,设计合理的测试用例,能够有效地揭示潜伏在软件里的缺陷,以便软件能够成功交付。

1.3 软件测试过程模型

软件测试是软件开发过程中的一个重要环节,是在软件投入运行前,对软件需求分析、设计规格说明和编码实现的最终审定,贯穿于软件定义与开发的整个过程中。

软件项目一旦开始,软件测试也随之开始。从测试过程可以看出,软件测试由一系列不同的测试阶段组成,即单元测试、集成测试、确认测试、系统测试和验收测试。软件开发是一个自顶向下逐步细化的过程,而软件测试是自底向上逐步集成的过程。低一级的测试为上一级的测试准备条件。测试过程的流程如图1.2所示。

1.3.1 单元测试

软件单元测试是检验程序的最小单位,也是软件设计的最小单位——模块有没有错误,它是在编码完成后必须进行的测试工作。一个单元应该有明确的功能定义、性能定义和接口定义,而且可以清晰地与其他单元区分开来。一个菜单、一个显示界面、一个类或者能够独立完成的具体功能都可以是一个单元。单元测试一般由程序开发者自行完成,因而单元

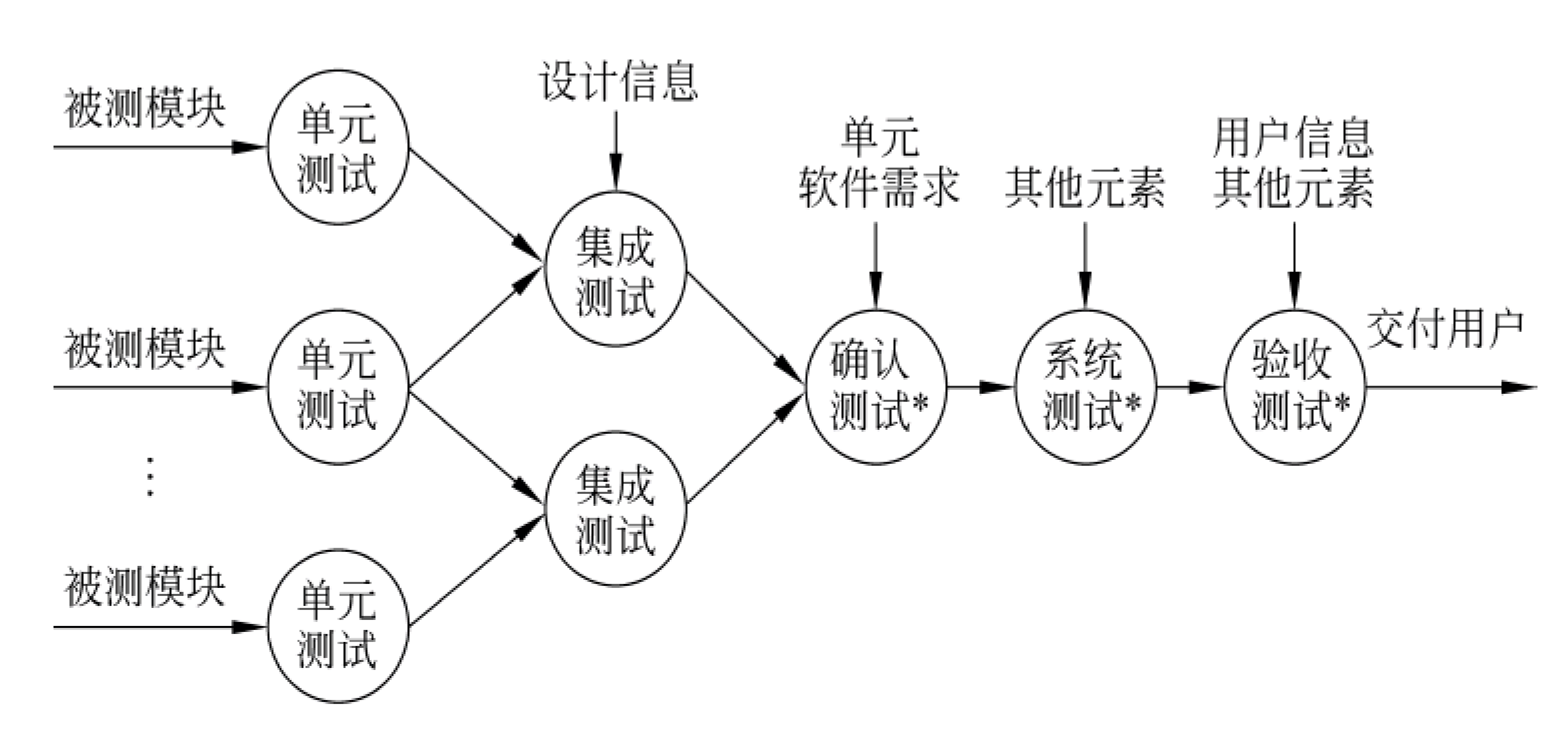


图 1.2 测试过程流程

测试大多是从程序内部结构出发设计测试用例的，即采用白盒测试方法。当有多个程序模块时，可并行独立开展测试工作。

1. 单元测试的目标

单元测试的主要目标就是通过测试，保证各个单元模块能够正确编码。单元测试除了保证测试代码的功能性，还需要保证代码在结构上具有可靠性和健壮性，能够在所有条件下正确响应。进行全面的单元测试，可以减少应用级别所需的工作量，并且彻底减少系统产生错误的可能性。如果手动执行，单元测试可能需要大量的工作，自动化测试会提高测试效率。只有完成了单元测试才能进入集成测试，所以单元测试是集成测试的基础。

2. 单元测试的内容

单元测试的主要内容有模块接口测试、局部数据结构测试、独立路径测试、错误处理测试和边界条件测试。这些测试都是作用在模块上的，共同完成单元测试的任务。

(1) 模块接口测试：是单元测试的基础。只有在数据能正确流入、流出的前提下，其他测试才有意义。因此，模块接口测试主要是对通过被测模块的数据流进行测试。为此，对模块接口，包括参数表、调用子模块的参数、全程数据和文件输入/输出操作都必须检查。

(2) 局部数据结构测试：检查局部数据结构是为了保证临时存储在模块内的数据在程序执行过程中的完整和正确。局部数据结构往往是错误的根源，应该仔细设计测试用例。设计测试用例时，要检查数据类型说明、初始化、默认值等方面的问题，还要查清全程数据对模块的影响。

(3) 独立路径测试：在一个模块中应该对每一条独立执行的路径进行测试，单元测试的基本任务是保证模块中的每条语句至少执行一次。选择适当的测试用例，对模块中重要的执行路径进行测试。对基本执行路径和循环进行测试是最常用且最有效的测试技术，可以发现大量路径错误。

(4) 错误处理测试：一个好的设计应该能够预见各种出错条件，并预设各种出错处理通道，进行错误处理测试。检查模块的错误处理功能是否包含有错误或缺陷。例如，是否拒绝不合理的输入，出错的描述是否难以理解，是否对错误定位有误，是否对出错原因报告有误，是否对错误条件的处理不正确，在对错误处理之前错误条件是否已经引起系统的干预，等等。

(5) 边界条件测试：是单元测试中最后也是最重要的一项任务。软件特别容易在边界

上失效，采用边界值分析技术，针对边界值及其左、右设计测试用例，很有可能发现新的错误。要特别注意数据流、控制流中刚好等于、大于或小于确定的比较值时出错的可能性。对这些地方要仔细地选择测试用例，认真加以测试。

此外，如果对模块运行时间有要求的话，还要专门进行关键路径测试，以确定最坏情况下和平均意义下影响模块运行时间的因素。这类信息对进行性能评价是十分有用的。

3. 单元测试的步骤

通常单元测试在编码阶段进行。在源程序代码编制完成，经过评审和验证，确认没有语法错误之后，就开始进行单元测试的测试用例设计。利用设计文档，设计可以验证程序功能、找出程序错误的多个测试用例。对于每一组输入，应有预期的正确结果。

模块并不是一个独立的程序，在考虑测试模块时，同时要考虑它和外界的联系，用一些辅助模块去模拟与被测模块相联系的其他模块。这些辅助模块分为两种。

(1) 驱动模块：相当于被测模块的主程序。它接收测试数据，把这些数据传送给被测模块，最后输出实测结果。

(2) 桩模块：用以代替被测模块调用的子模块。桩模块可以做少量的数据操作，不需要把子模块的所有功能都带进来，但不允许什么事情也不做。

被测模块、与它相关的驱动模块及桩模块共同构成了一个"测试环境"。

如果一个模块要完成多种功能且以程序包或对象类的形式出现，例如 Ada 中的包、Modula 中的模块以及 C++ 中的类。这时可以将这个模块看成由几个小程序组成。对其中的每个小程序先进行单元测试要做的工作，对关键模块还要做性能测试。对支持某些标准规程的程序，更要着手进行互联测试。有人把这种情况特别称为模块测试，以区别单元测试。

4. 面向对象的单元测试

传统的单元测试是针对程序的函数、过程或者完成某一特定功能的程序块完成的。面向对象的单元测试对象是软件设计的最小单位——类。单元测试的依据是详细设计，单元测试应该对类中所有的重要属性和方法设计测试用例，以便发现类内部的错误。系统内的多个类可以并行地进行测试，采用白盒测试技术。测试类成员函数时，可以沿用传统的单元测试中的测试方法，如等价分类法、因果图法、边值分析法、逻辑覆盖法和路径分析法等。

1) 单元测试的内容

单元测试实际上就是对类的测试。类测试的目的就是确保类代码能够完全满足类说明所描述的要求。对一个类的测试就是确保它只做规定的事情。每个类封装了属性和方法。方法就是管理这些数据的操作，一个类中包含许多不同的操作，一个特殊的操作也可以出现在许多不同的类中，而不是个体的模块。传统的单元测试只能测试一个操作，即功能。但是在面向对象的单元测试中，一个操作功能只能作为一个类的一部分，不是独立的，类中有多个操作，就要进行多个操作的测试。另外，父类中定义的某个操作被多个子类继承，不同子类中某个操作在使用时又有区别，所以还必须对每个子类中的某个操作进行测试。对类的测试强调对语句应该有100%的执行代码覆盖率。在运行了各种类型的测试用例后，如果代码的覆盖不完整，就可能意味着该类包含了额外的文档支持的行为，需要增加更多的测试用例来进行测试。

2）方法的测试

类的行为是通过其内部方法来表现的，方法可以看作传统测试中的模块，因此传统针对模块测试案例的技术，都可以作为测试类中每个方法的主要技术。在面向对象技术中，为了提高方法的重用性，每个方法所实现的功能应尽量小，每个方法常常只由几行代码组成，控制比较简单，因此测试用例的设计相对比较容易。在面向对象系统中的方法是通过消息驱动执行的。要测试类中的方法，必须用一个驱动程序对被测方法发一条消息以驱动其执行，如果被测模块或方法中调用了其他的模块或方法，则都需要设计一个模拟被调子程序功能的存根程序。驱动程序、存根程序及被测模块或方法组成一个独立的可执行的单元。

方法测试中有两个方面要加以注意。首先，方法执行的结果并不一定返回调用者，有的可能是改变被测对象的状态，例如类中所有的属性值。状态是外界不可见的，为了测试对象状态是否已经发生变化，在驱动程序中还必须给对象发送一些额外的信息。其次，除了类中自己定义的方法，还可能存在从基类继承来的方法，这些方法虽然在基类中已经测试过，但派生类往往需要再次测试。

另外，在面向对象软件中，在保证单个方法功能正确的基础上，还应该测试方法之间的协作关系。操作被封装在类中，对象彼此间通过发送消息启动相应的操作。但是，对象并没有明显地规定用什么次序启动它的操作才是合法的。这时，对象就像一个有多个入口的模块，因此，测试方法必须依不同次序组合的情况进行。测试完全的次序组合通常是不可能的，在设计测试用例时，可以从各种次序组合中选出最可能发现属性和操作错误的若干种情况，使用等价类划分、边界值和错误推测等技术着重进行测试。其测试步骤与单个方法的测试步骤类似。

总而言之，以方法作为单元进行测试，传统的方法都可以使用。在设计测试用例时，必须提供能够实例化的桩类，以及起驱动器作用的"主程序"类，来提供和分析测试用例。

1.3.2 集成测试

实践证明，一些模块虽然能够单独工作，但并不能保证连接起来也能正常的工作。程序在某些局部反映不出来的问题，在全局上很可能暴露出来，影响功能的实现。因此，单元测试完成后，必须进行集成测试。

集成测试，也叫组装测试或联合测试，是在单元测试的基础上进行的。单元测试以后，将所有模块按照设计的要求(如根据结构图)集成为子系统或系统，进行集成测试。

1. 集成测试的目标

集成测试的目标是按照设计的要求使用那些通过单元测试的构件来构造程序结构。单个模块具有高质量，但不足以保证整个系统的质量。有许多隐蔽的失效是高质量模块间发生非预期交互而产生的。

集成测试要考虑以下问题：

(1) 在把各个模块连接起来的时候，穿越模块接口的数据是否会丢失；

(2) 各个子功能组合起来，能否达到预期要求的父功能；

(3) 一个模块的功能是否会对另一个模块的功能产生不利的影响；

(4) 全局数据结构是否有问题；

(5) 单个模块的误差积累起来，是否会放大，从而达到不可接受的程度。

要想发现并排除在模块连接中可能发生的上述问题，就需要进行集成测试。

2. 集成测试的特点

集成测试的特点如下：

(1) 单元测试对于模块间接口信息内容的正确性、相互调用关系是否符合设计无能为力，只能靠集成测试来进行保障。

(2) 集成测试用例从程序结构出发，目的性、针对性更强，测试发现问题的效率更高，定位问题的效率也较高。

(3) 能够较容易地测试到系统测试用例难以模拟的特殊异常流程，从纯理论的角度来讲，集成测试能够模拟所有实际情况。

(4) 集成测试具有可重复性强、对测试人员透明的特点，发现问题后容易定位，能够有效地加快进度，减少隐患。

3. 集成测试的方案

集成测试首先要制订测试计划，然后再进行设计，设计完成后进行实施和执行，最后进行评估。

在测试过程中，组合模块一般有两种不同的集成方式：一次性集成方式和增量式集成方式。

1) 一次性集成方式

一次性集成方式是一种非增量集成方式，也叫做整体拼装。按这种集成方式，首先对每个模块分别进行模块测试，然后再把所有模块集成在一起进行测试，最终得到要求的软件系统。

2) 增量式集成方式

增量式集成方式也称为递增集成法，即逐次将未曾测试的模块和已测试的模块(或子系统)结合成程序包，然后将这些模块集成为较大系统，在集成的过程中边连接边测试，以发现连接过程中产生的问题。最后逐步集成为要求的软件系统。

根据集成的过程，又可以分为自顶向下集成和自底向上集成。

自顶向下集成是将模块按照系统的程序结构，沿控制层次自顶向下进行集成。自顶向下的集成方式有很多优点：能够在测试过程中较早地验证主要的控制和断点；可以首先实现和验证一个完整的软件功能，可先对逻辑输入的分支进行集成和测试，检查和克服潜藏的错误和缺陷；功能可行性较早得到证实，给开发者和用户带来成功的信心。

自底向上的集成方式是从程序模块结构的最底层的模块开始集成和测试。因为模块是自底向上进行集成，对于一个给定的模块，它的子模块(包括子模块的所有下属模块)已经集成并测试完成，所以不再需要桩模块。自底向上集成的步骤如下：

(1) 由驱动模块控制最底层模块的并行测试，也可以把最底层模块组合成实现某一特定软件功能的簇，由驱动模块控制它进行测试。

(2) 用实际模块代替驱动模块，与它已测试的直属子模块集成为子系统。

(3) 为子系统配备驱动模块，进行新的测试。

(4) 判断是否已集成到达主模块，是否结束测试，否则执行(2)。

自顶向下集成的方式和自底向上集成的方式各有优缺点。一般来讲，一种方式的优点是另一种方式的缺点。

(1) 自顶向下集成方式的缺点是需要建立桩模块。要使桩模块模拟实际子模块的功能十分困难,涉及复杂算法,真正输入/输出的模块一般在底层,它们是最容易出问题的模块,到测试和集成的后期才遇到这些模块,一旦发现问题导致过多的回归测试。

这种方式的优点是能够较早地发现在主要控制方面的问题。

(2) 自底向上集成方式的缺点是"程序一直未能作为一个实体存在,直到最后一个模块加上去后才形成一个实体"。就是说,在自底向上集成和测试的过程中,对主要的控制直到最后才接触到。

这种方式的优点是不需要桩模块,而建立驱动模块一般比建立桩模块容易,同时由于涉及复杂算法和真正输入/输出的模块最先得到集成和测试,可以把最容易出问题的部分在早期解决。此外,自底向上集成的方式可以实施多个模块的并行测试,提高测试效率。

混合集成式测试就是把以上两种方式结合起来进行集成和测试,这样可以兼具两者的优点。混合式集成方式有衍变的自顶向下的集成方式、自底向上-自顶向下的集成测试和回归测试3种方式。

衍变的自顶向下的集成方式的基本思想是强化对输入/输出模块和引入新算法模块的测试,并自底向上集成为功能相当完整且相对独立的子系统,然后由主模块开始自顶向下进行集成测试。

自底向上-自顶向下的集成测试首先对含读操作的子系统自底向上直至根结点模块进行集成和测试,然后对含写操作的子系统作自顶向下的继承与测试。

回归测试采取自顶向下的方式测试所修改的模块及其子模块,然后将这一部分视为子系统,再自底向上测试,以检查该子系统与其上级模块的接口是否匹配。

在继承测试时,测试者应当确定关键模块,对这些关键模块及早进行测试。关键模块至少应具有以下几种特征之一:

(1) 满足某些软件需求。

(2) 在程序的模块结构中位于较高的层次(高层控制模块)。

(3) 较复杂、较易发生错误。

(4) 有明显定义的性能要求。

在做回归测试时,也应该集中测试关键模块的功能。

1.3.3 确认测试

确认测试的目的是向未来的用户表明系统能够像预定的要求那样进行工作。经过集成测试,已经按照设计把所有的模块组装成了一个完整的软件系统,接口错误也已经基本排除了,接着就应该进一步验证软件的有效性,这就是确认测试的任务,即软件的功能和性能如同用户所期待的那样。

确认测试又称有效性测试。有效性测试是在模拟的环境下,运用黑盒测试的方法,验证被测软件是否满足需求规格说明书列出的需求。其任务是验证软件的功能和性能及其他特性是否与用户的要求一致。对软件的功能和性能要求在软件需求规格说明书中已经明确规定,它包含的信息就是软件确认测试的基础。

1. 确认测试的内容

确认测试包括以下9个内容。

(1) 安装测试：确保该软件在正常情况和异常情况的不同条件下，在软件安装后都可以立即正常运行。

(2) 功能测试：对产品的各个功能进行验证，根据功能测试用例，逐项测试，检查产品是否达到了用户要求的功能。

(3) 可靠性测试：根据软件系统可靠性结构、寿命类型和各单元的可靠性试验信息，利用概率统计方法，评估出系统的可靠性特征量。

(4) 安全性测试：验证应用程序的安全服务和识别潜在安全性缺陷的过程。

(5) 时间及空间性能测试：通过自动化的测试工具模拟多种正常、峰值以及异常负载条件来对系统的各项性能指标进行测试。

(6) 易用性测试：易用性指用户使用软件时是否感觉方便，比如是否最多点击鼠标三次就可以达到用户的目的。

(7) 可移植性测试：测试软件是否可以被成功移植到指定的硬件或软件平台上。

(8) 可维护性测试：一个软件系统或组件可以被修改的容易程度，这个修改一般是因为缺陷纠正、性能改进或特性增加引起的。

(9) 文档测试：检验样品用户文档的完整性、正确性、一致性、易理解性和易浏览性。

2. 确认测试的方法

1) 确认测试标准

实现软件确认要通过一系列黑盒测试。确认测试同样需要制订测试计划和过程，测试计划应规定测试的种类和测试进度，测试过程则定义一些特殊的测试用例，旨在说明软件与需求是否一致。无论是计划还是过程，都应该着重考虑软件是否满足合同规定的所有功能和性能，文档资料是否完整、准确，人机界面和其他方面(例如可移植性、兼容性、错误恢复能力和可维护性等)是否令用户满意。

确认测试的结果有两种可能：一种是功能和性能指标满足软件需求说明的要求，用户可以接受；另一种是软件不满足软件需求说明的要求，用户无法接受。项目进行到这个阶段才发现严重错误和偏差，一般很难在预定的工期内改正，因此必须与用户协商，寻求一个妥善解决问题的方法。

2) 配置复审

确认测试的另一个重要环节是配置复审。复审的目的在于保证软件配置齐全、分类有序，并且包括软件维护所必需的细节。

3) α、β 测试

事实上，软件开发人员不可能完全预见用户实际使用程序的情况。例如，用户可能错误地理解命令，或提供一些奇怪的数据组合，亦可能对设计者自己知道的输出信息迷惑不解，等等。因此，软件是否真正满足最终用户的要求，应由用户进行一系列“验收测试”。验收测试既可以是非正式的测试，也可以有计划、有系统地测试。有时，验收测试长达数周甚至数月，不断暴露错误，导致开发延期。一个软件产品可能拥有众多用户，不可能由每个用户验收，此时多采用称为 α、β 测试的过程，以期发现那些似乎只有最终用户才能发现的问题。

α 测试是指软件开发公司组织内部人员模拟各类用户对即将面市的软件产品(称为 α 版本)进行测试，试图发现错误并修正。α 测试的关键在于尽可能逼真地模拟实际运行环境和用户对软件产品的操作，并尽最大努力涵盖所有可能的用户操作方式。在该阶段中，需

要准备β测试的测试计划和测试用例。在软件开发周期中，根据功能性特征，所需的α测试的次数应在项目计划中规定。

β测试是指软件开发公司组织各方面的典型用户在日常工作中实际使用经过α测试调整的软件产品，即β版本，并要求用户报告异常情况，提出批评意见。

β测试是一种现场测试，一般由多个客户在软件真实运行环境下实施，因此开发人员无法对其进行控制。β测试的主要目的是评价软件技术内容，发现任何隐藏的错误和边界效应。它还要对软件是否易于使用以及用户文档初稿进行评价，发现错误并进行报告。β测试也是一种详细测试，需要覆盖产品的所有功能点，因此依赖于功能性测试。在测试阶段开始前应准备好测试计划，清楚列出测试目标、范围、执行的任务，以及描述测试安排的测试矩阵。客户对异常情况进行报告，并将错误在内部进行文档化以供测试人员和开发人员参考。

1.3.4 系统测试

系统测试是将经过集成测试的软件作为系统计算机的一个部分，与系统中其他部分结合起来，在实际运行环境下对计算机系统进行的一系列严格、有效的测试，以便发现软件潜在的问题，保证系统的正常运行。

1. 系统测试的目标

系统测试必须达到以下几个目标：确保系统测试的活动是按计划进行的；验证软件产品是否与系统需求用例不相符合或与之矛盾；建立完善的系统测试缺陷记录跟踪库；确保软件系统测试活动及其结果及时通知相关小组和个人。

2. 系统测试的内容

系统测试是将经过集成测试的软件作为系统计算机的一个部分，与系统中其他部分结合起来，在实际运行环境下对计算机系统进行的一系列严格、有效的测试，以发现软件潜在的问题，保证系统的正常运行。

系统测试的目的是验证最终软件系统是否满足用户的需求。

系统测试的主要内容如下：

(1) 功能测试。即测试软件系统的功能是否正确，其依据是需求文档，如《产品需求规格说明书》。由于正确性是软件最重要的质量因素，所以功能测试必不可少。

(2) 健壮性测试。即测试软件系统在异常情况下能否正常运行的能力。健壮性有两层含义：一是容错能力，二是恢复能力。

3. 系统测试的分类

比较常见的、典型的系统测试包括恢复测试、安全测试和压力测试。

1) 恢复测试

恢复测试作为一种系统测试，主要关注导致软件运行失败的各种条件，并验证其恢复过程能否正确执行。在特定情况下，系统需具备容错能力。另外，系统失效必须在规定时间段内被更正，否则将会导致严重的经济损失。

2) 安全测试

安全测试用来验证系统内部的保护机制，以防止非法侵入。在安全测试中，测试人员扮演试图侵入系统的角色，采用各种办法试图突破防线。因此在制定系统安全设计的准则时要想方设法使侵入系统所需的代价更加昂贵。

3）压力测试

压力测试是指在正常资源下使用异常的访问量、频率或数据量来执行系统测试。在压力测试中可执行以下测试：

(1) 如果平均中断数量是每秒一到两次，那么设计特殊的测试用例产生每秒10次中断。

(2) 输入数据量增加一个量级，确定输入功能将如何响应。

(3) 在虚拟操作系统下，产生需要最大内存量或其他资源的测试用例，或产生需要过量磁盘存储的数据。

4. 系统测试的步骤

系统测试包括以下4个步骤。

(1) 制定系统测试计划。

系统测试小组各成员共同协商测试计划。测试组长按照指定的模板起草系统测试计划。该计划主要包括测试范围、测试方法、测试环境与辅助工具、测试完成准则和人员及任务表。系统测试计划由项目经理审批，批准后，转向步骤(2)。

(2) 设计系统测试用例。

系统测试小组各成员依据系统测试计划和指定的模板，设计系统测试用例，测试组长邀请开发人员和同行专家对系统测试用例进行技术评审。该测试用例通过技术评审后，转向步骤(3)。

(3) 执行系统测试。

系统测试小组各成员依据系统测试计划和系统测试用例执行系统测试。将测试结果记录在系统测试报告中，用缺陷管理工具来管理所发现的缺陷，并及时通报给开发人员。

(4) 缺陷管理与改错。

从前3个步骤中，任何人发现软件系统中的缺陷时都必须使用指定的缺陷管理工具。该工具将记录所有缺陷的状态信息，并自动产生缺陷管理报告。开发人员及时消除已经发现的缺陷，然后马上进行回归测试，以确保不会引入新的缺陷。

1.3.5 验收测试

验收测试是在软件产品完成了功能测试和系统测试之后，产品发布之前所进行的软件测试活动。它是技术测试的最后一个阶段，也称为交付测试。验收测试的目的是确保软件准备就绪，可以让最终用户使用其既定功能和任务。

1. 验收测试的内容

验收测试的内容通常可以包括安装(升级)、启动与关机、功能测试(正例、重要算法、边界、时序、反例和错误处理)、性能测试(正常的负载、容量变化)、压力测试(临界的负载、容量变化)、配置测试、平台测试、安全性测试、恢复测试(在出现掉电、硬件故障或切换、网络故障等情况时，系统是否能够正常运行)和可靠性测试等。

性能测试和压力测试一般情况下是在一起进行的，通常还需要辅助工具的支持。在进行性能测试和压力测试时，测试范围必须限定在那些使用频度高的和时间要求苛刻的软件功能子集中。由于开发方已经事先进行过性能测试和压力测试，因此可以直接使用开发方的辅助工具。也可以购买或自己开发辅助工具。

如果执行了所有的测试案例、测试程序或脚本，用户验收测试中发现的所有软件问题都已解决，而且所有的软件配置均已更新和审核，可以反映出软件在用户验收测试中所发生的变化，用户验收测试就完成了。

2. 验收测试的步骤

验收测试包括以下5个步骤。

(1) 建立测试计划。测试计划在需求分析阶段建立，主要了解软件功能和性能要求、软硬件环境要求等，并特别要了解软件的质量要求和验收要求。根据软件需求和验收要求编制测试计划，制定需测试的测试项，制定测试策略及验收通过准则，并通过有客户参与的计划评审。

(2) 建立测试环境。根据验收测试计划、项目或产品验收准则完成测试用例的设计，并通过评审。

(3) 准备测试数据，执行测试用例，记录测试结果。

(4) 分析测试结果。根据验收通过准则分析测试结果，做出验收是否通过的结论并给出测试评价。通常会有4种情况：测试项目通过；测试项目没有通过，并且不存在变通方法，需要作大量的修改；测试项目没有通过，但是存在变通方法，在维护后期或下一个版本改进；测试项目无法评估或者无法给出完整的评估，此时必须给出原因。如果是因为该测试项目没有说清楚，应该修改测试计划。

(5) 提交测试报告。根据产品设计说明书、详细设计说明书、验收测试结果和发现的错误信息，评价系统的设计与实现，最终通过验收测试报告和缺陷报告等体现出来。

3. 验收测试的常用策略

实施验收测试的常用策略有3种，它们分别是：正式验收、非正式验收以及β测试。一般选择什么样的验收策略，主要根据合同需求、组织和公司标准以及应用领域来决定。

1) 正式验收测试

正式验收测试是一项管理严格的过程，它通常是系统测试的延续。计划和设计这些测试的周密和详细程度不亚于系统测试。选择的测试用例应该是系统测试中所执行测试用例的子集。不要偏离所选择的测试用例方向，这一点很重要。在很多组织中，正式验收测试是完全自动执行的。

这种测试形式的优点是：要测试的功能和特性都是已知的；测试的细节是已知的并且可以对其进行评测；这种测试可以自动执行，支持回归测试；可以对测试过程进行评测和监测；可接受性标准是已知的。

其缺点包括：要求大量的资源和计划；这些测试可能是系统测试的再次实施；由于只查找预期的缺陷，可能无法发现软件中由于主观原因而造成的缺陷。

2) 非正式验收测试

在非正式验收测试中，执行测试过程的限定不像正式验收测试中那样严格。在此测试中，确定并记录要研究的功能和业务任务，但没有可以遵循的特定测试用例。测试内容由各测试员决定。这种验收测试方法不像正式验收测试那样组织有序，相对主观。

大多数情况下，非正式验收测试是由最终用户组织执行的。

这种测试形式的优点是：要测试的功能和特性都是已知的；可以对测试过程进行评测和监测；可接受性标准是已知的；与正式验收测试相比，可以发现更多由于主观原因造成的

缺陷。

其缺点包括：要求资源、计划和管理资源；无法控制所使用的测试用例；最终用户可能沿用系统工作的方式，并可能无法发现缺陷；最终用户可能专注于比较新系统与遗留系统，而不是专注于查找缺陷；用于验收测试的资源不受项目的控制，并且可能受到压缩。

3）β测试

在这3种验收测试策略中，β测试需要的控制是最少的。在β测试中，采用的细节多少、数据和方法完全由各测试员决定。各测试员负责创建自己的环境，选择数据，并决定要研究的功能、特性或任务。各测试员负责确定自己对于系统当前状态的接受标准。

β测试由最终用户实施，通常开发（或其他非最终用户）组织对其的管理很少或不进行管理。β测试是所有验收测试策略中最主观的。

这种测试形式的优点是：测试由最终用户实施；大量的潜在测试资源；提高客户对参与人员的满意程度；与正式或非正式验收测试相比，可以发现更多由于主观原因造成的缺陷。

其缺点包括：未对所有功能和/或特性进行测试；测试流程难以评测；最终用户可能沿用系统工作的方式，并可能没有发现或没有报告缺陷；最终用户可能专注于比较新系统与遗留系统，而不是专注于查找缺陷；用于验收测试的资源不受项目的控制，并且可能受到压缩；可接受性标准是未知的。

1.3.6 测试模型

软件测试和软件开发一样，都遵循软件工程原理和管理学原理。因此，软件测试也有很多的测试模型，这些模型将测试活动进行了抽象，明确了测试和开发之间的关系，是测试管理的重要参考依据。

1. V模型

在软件测试方面，V模型是最广为人知的模型。V模型已存在了很长时间，和瀑布开发模型有着一些共同的特性，由此也和瀑布模型一样地受到了批评和质疑。V模型中的过程从左到右，描述了基本的开发过程和测试行为。V模型的价值在于它非常明确地标明了测试过程中存在的不同级别，并且清楚地描述了这些测试阶段和开发过程期间各阶段的对应关系。

其局限性是：把测试作为编码之后的最后一个活动，需求分析等前期产生的错误直到后期的验收测试才能发现。

2. W模型

W模型由Evolutif公司提出，相对于V模型，W模型更科学。W模型是V模型的发展，强调的是测试伴随着整个软件开发周期，而且测试的对象不仅仅是程序，需求、功能和设计同样要测试。测试与开发是同步进行的，从而有利于尽早地发现问题。

W模型也有局限性。W模型和V模型都把软件的开发视为需求、设计、编码等一系列串行的活动，无法支持迭代、自发性以及变更调整。

3. X模型

X模型是针对单独程序片段所进行的相互分离的编码和测试，此后将进行频繁的交接，通过集成最终成为可执行的程序，然后再对这些可执行程序进行测试。已通过集成测试的

成品可以进行封装并提交给用户，也可以作为更大规模和范围内集成的一部分。X 模型还定位了探索性测试，这是不进行事先计划的特殊类型的测试，这一方式往往能帮助有经验的测试人员在测试计划之外发现更多的软件错误。但这样可能对测试造成人力、物力和财力的浪费，对测试员的熟练程度要求比较高。

4. H 模型

H 模型中，软件测试过程活动完全独立，贯穿于整个产品的周期，与其他流程并发地进行。某个测试点准备就绪时，就可以从测试准备阶段进行到测试执行阶段。软件测试可以尽早进行，并且可以根据被测物的不同而分层次进行。

H 模型揭示了一个原理：软件测试是一个独立的流程，贯穿产品的整个生命周期，与其他流程并发地进行。H 模型指出软件测试要尽早准备，尽早执行。不同的测试活动可以是按照某个次序先后进行的，但也可能是反复的，只要某个测试达到准备就绪点，测试执行活动就可以开展。

对上述 4 种测试模型可以总结如下：

V 模型——非常明确地标注了测试过程中存在的不同类型的测试。

W 模型——非常明确地标注了生产周期中开发与测试之间的对应关系。

X 模型——这个模型指出整个测试过程是在探索中进行的。

H 模型——软件测试是一个独立的流程，贯穿产品的整个生命周期，与其他流程并发地进行。

1.4 软件测试职业发展和现状

1.4.1 软件测试的现状

目前，越来越多的软件开发机构认识到测试的重要性，成立了相应的测试机构，如 QA 小组，并且配有专门的测试人员。但是从整体上来说，对测试的认识程度仍旧不够，还存在以下的问题：

(1) 测试工作滞后。

大多数错误都是在编码阶段产生的，但也可能出现在整个项目生命周期中的其他地方。应该尽早发现错误，因为每迟一步，发现错误所需时间就会加长，消除错误的成本就会增加。

(2) 缺乏合适的测试方法。

测试方法应与开发生命周期相适应。测试需要有序、有步骤地进行，需要明确测试的目标范围，确定如何测试，建立测试程序，审查测试结果。保证应用的各个方面都被测试覆盖，从而保证应用的质量。

(3) 测试人员的配备不合理。

没有专门的测试人员，设计和测试不分开，对现有项目和未来项目缺乏分析，不能预计所需人员。人员也没有很好地培训。

(4) 缺乏测试技术。

缺乏测试工具，忽视不同类型的测试。随着软件设计的复杂化，开发及分发软件所使用

的技术，如图形用户界面、分布式处理、庞大的分散网络和 Web 技术等的更新，通过手工方式实现测试较为困难，软件测试面临新的机遇，软件测试的实施需要一种明确的方法和所需的自动支持。

(5) 领导层和项目经理对测试缺乏正确认识。

测试经费不足，导致测试工作难以顺利进行，造成成本急剧上升、测试不完全等后果。

总而言之，要成功开发出高质量的软件产品，提高企业的竞争力，必须改变上述现状，重视并加强测试的工作，只有充分测试才能确保软件的质量。

1.4.2 软件测试的职业发展

软件测试给各方面都带来了机遇，提供了很多和软件测试相关的职业。软件测试的职业发展主要体现在以下几个方面。

(1) 出现了软件测试工作岗位。

目前，产生了许多与软件测试相关的工作岗位以及与测试相关的商机，如外包、培训等。

(2) 出现软件测试研究工作。

有些人开始专门研究软件测试，研究如何把测试理论和实际相结合，对测试方法和测试工具进行研究，并建立测试实验室。

(3) 软件测试教育体系逐步形成。

现在还没有专门的测试学位，但这是一个趋势，因为软件测试与软件是密切相关的，软件测试专业的教育体系的形成需要一个过程，需要有专门的教程和专门的实习，以提高软件测试教育的水平。

(4) 成为政府关注的新领域。

政府也越来越关注软件测试的工作，在行业中和高校中支持和主办测试研究，开始考虑统一的软件质量度量标准，通过培训和服务壮大产业规模，设立测试实验室以改进质量，形成基准。

(5) 给个人提供了大量工作的机会。

软件测试可以给个人提供大量工作的机会，主要分为技术上和管理上两大类。技术性职位包括软件测试工程师、软件开发工程师、软件测试技术主管和软件测试设计师；管理性职位包括测试主管、测试管理者和项目主管。

总而言之，软件测试前途无量，将成为中国软件产业发展的动力，软件工业的向前发展离不开软件测试。

1.5 本章小结

在本章中，首先介绍了计算机软件可靠性问题，然后介绍了软件测试基本知识，包括软件测试背景、软件测试的原则以及软件测试的分类。在软件测试过程的模型中，介绍了在整个软件测试过程中的各个步骤，并介绍了 4 种不同的测试模型，最后分析了软件测试的现状，并提出了软件测试的职业发展。

习　　题

1. 软件测试的原则是什么?
2. 如何从不同角度对软件测试进行分类?
3. 简述软件测试的整个过程。
4. 测试的模型有哪几种?各自的优缺点是什么?

第2章　程序示例

本章主要使用两个程序示例来描述各种单元测试方法,这两个实例分别是逻辑结构比较复杂的 NextDate 函数和 ATM(自动取款机)系统。其中,ATM 系统采用了面向对象的方法——统一建模语言(UML)进行描述。

2.1　通用伪代码

人们在用不同的编程语言实现同一个算法时意识到,它们的实现(注意:这里是实现,不是功能)会大不相同。尤其是对于那些熟悉不同编程语言的程序员要理解一个用其他编程语言编写的程序的功能时可能很困难,因为程序语言的形式限制了程序员对程序关键部分的理解,这样伪代码就应运而生了。伪代码提供了更多的设计信息。

伪代码(pseudocode)是一种算法描述语言,它是一种非正式的,类似于英语结构的,用于描述模块结构图的语言。使用伪代码的目的是使被描述的算法可以容易地以任何一种编程语言(Pascal、C、Java 等)实现。因此,伪代码必须结构清晰、代码简单、可读性好,并且类似自然语言,介于自然语言与编程语言之间。以编程语言的书写形式指明算法职能。使用伪代码,不用拘泥于具体实现。相比于程序语言(例如 Java、C++、C、Delphi 等),伪代码更类似于自然语言。它是半形式化、不标准的语言,可以将整个算法运行过程的结构用接近自然语言的形式(可以使用任何一种熟悉的文字,关键是把程序的意思表达出来)描述出来。

【例】　输入 3 个数,打印输出其中最大的数。可用如下的伪代码表示:

```
BEGIN(算法开始)
    输入 A,B,C
    IF A>B
        THEN A→Max
    ELSE
        B→Max
    IF C>Max
        THEN C→Max
    PRINT Max
END (算法结束)
```

2.2　伪代码的语法规则

在伪代码中,每一条指令占一行(else if 例外),指令后不跟任何符号(Pascal 和 C 中语句要以分号结尾);书写上的缩进表示程序中的分支程序结构。这种缩进风格也适用于 if-then-else 语句。用缩进取代传统 Pascal 中的 begin 和 end 语句以及 C 和 Java 中的{和}来表示程序的块结构,可以大大提高代码的清晰性;同一模块的语句有相同的缩进量,次一级

模块的语句相对于其上一级模块的语句缩进一级。

伪代码的一些基本格式如下：

(1) 开始和结束的表示法：BEGIN(表示开始)，END(表示结束)。

(2) 赋值语句用符号←表示，如赋值用 x←y 表示将 y 的值赋给变量 x，其中 x 是一个变量，y 是与 x 同类型的一个变量或表达式。

(3) 多重赋值 i←j←e 是将表达式 e 的值赋给变量 i 和 j，这种表示与 j←e 和 i←e 等价。

(4) 在伪代码中，变量名和保留字不区分大小写，这一点和 Pascal 相同，但与 C、C++ 或 Java 语言不同。

(5) 循环语句有 3 种：while 循环、repeat-until 循环(相当于 do-while)和 for 循环。例如：

```
while i<=20
```

伪代码示例：

```
x ← 0
y ← 0
z ← 0
while x <N
    do x ← x +1
      y ← x +y
      for t ← 0 to 10
        do z ← ( z +x * y ) / 100
          repeat
            y ← y +1
            z ← z - y
          until z <0
        z ← x * y
    y ← y / 2
```

上述伪代码的 Java 实现如下：

```
x =0;
y =0;
z =0;
while( x <N ){
  x ++;
  y =x+y;
  for( t =0; t <10; t++)
  {
    z =( z +x * y ) / 100;
    do {
        y ++;
        z =z-y;
    } while( z >=0 );
  }
```

```
  z =x * y;
}
y=y/ 2;
```

2.3 NextDate 程序

NextDate 程序是软件测试中的一个非常经典的例子，该程序主要体现了输入变量之间的复杂的逻辑关系。

2.3.1 问题描述

NextDate 问题是一个有 3 个输入变量 day、month 和 year 的函数，输入这 3 个变量，输出为当前输入后一天的日期。这里 day、month 和 year 分别取整数，且应满足如下要求：

(1) 1≤day≤31

(2) 1≤month≤12

(3) 1900≤year≤2015

设定输入变量的基本条件后，还应该给出更严格的说明，即对于 day、month 和 year 输入无效的取值的情况说明，如果这 3 个变量中的任何一个不满足以上的 3 个条件中的任何一个，则应该给出提示信息 Invalid Input Date。

2.3.2 NextDate 程序分析

NextDate 程序的逻辑结构复杂的原因主要来自两方面：第一，输入域的复杂性；第二，判断某年是否闰年的复杂性。其中，对于闰年的判断可以采用如下方法：如果年份可以被 4 整除，并且不是整世纪年(即不能被 100 整除)；或年份可以被 400 整除。两个条件其中一个满足即可判定该年为闰年。如 1996 年可以被 4 整除但不能被 100 整除，2000 年可以被 400 整除，因此为闰年；但 1900 年可以被 4 整除，也可以被 100 整除，因此该年不是闰年。

2.3.3 NextDate 程序实现

下面的 NextDate 函数采用 Java 语言实现。

```
String Date(int d,int m,int y){
    switch(m){
              case 1:case 3:case 5:case 7:case 8:case 10: {
                 if (d <31) {
                     nextday =d +1;
                     nextmonth =m;
                 } else {
                     nextday =1;
                     nextmonth =m +1;
                 }
                 nextyear=y;
              }
              break;
```

```
                case 4:
                case 6:
                case 9:
                case 11: {
                    if (d < 30) {
                        nextday = d + 1;
                        nextmonth = m;
                    } else {
                        nextday = 1;
                        nextmonth = m + 1;
                    }
                    nextyear= y;
                }
                break;
                case 12: {
                    if (d < 31) {
                        nextday = d + 1;
                        nextmonth = m;
                        nextyear= y;
                    } else {
                        nextday = 1;
                        nextmonth = 1;
                        nextyear= y+ 1;
                    }
                }break;
                case 2: {
                    if (d < 28) {
                        nextday = d + 1;
                        nextmonth = m;
                    } else {
                        if (d == 28) {
                            if (y % 4 == 0 && y % 100 != 0 || y % 400 == 0) {//闰年
                                nextday = d + 1;
                                nextmonth = m;
                            } else {
                                nextday = 1;
                                nextmonth = 3;
                            }
                        }
                    }
                    nextyear= y;
                }break;
            }
        return nextday+ "/"+ nextmonth+ "/"+ nextyear;
    }
```

2.4 UML 语言

UML(Unified Modeling Language,统一建模语言)融合了 Booch、OMT 和 OOSE 方法中的基本概念,是一种面向对象的建模语言。它可以帮助用户对软件系统进行面向对象的描述和建模。简单地说,UML 由 3 个内容组成,分别是事物、关系和图。UML 通过建立类与类之间的关系以及类/对象如何相互配合实现系统的动态行为。标准建模语言 UML 的重要内容可以由下列 5 类图(9 种图形)来定义。

1. 用例图

用例图是从用户角度描述系统功能,并指出各功能的操作者。简单地说,用例图就是软件产品外部特性描述的视图。用例图站在用户的角度,描述软件产品的需求,分析产品所需的功能和动态行为。

2. 静态图

静态图是其他视图的基础,用于对应用领域中的概念以及系统实现有关的内部概念建模。静态图将实体看作被类所指定、拥有并使用的物体。

静态图包括类图、对象图和包图。

类图描述系统中类的静态结构。不仅定义系统中的类,表示类之间的联系如关联、依赖、聚合等,也包括类的内部结构(类的属性和操作)。类图描述的是一种静态关系,在系统的整个生命周期都是有效的。

对象图是类图的实例,几乎使用与类图完全相同的标识。两者的不同点在于对象图显示类的多个对象实例,而不是实际的类。由于对象存在生命周期,因此对象图只能在系统某一时间段存在。

包图由包或类组成,表示包与包之间的关系。用于描述系统的分层结构。

3. 行为图

行为图描述系统的动态模型和组成对象间的交互关系。行为图包括状态图和活动图。

状态图能够描述类的对象所有可能的状态以及事件发生时状态的转移条件。状态图是对类图的一种补充。而实际上并不需要为所有的类画状态图,仅为那些有多个状态,其行为受外界环境的影响并且发生改变的类画状态图。

活动图描述满足用例要求所要进行的活动以及活动间的约束关系,有利于识别并行活动。

4. 交互图

交互图描述对象间的交互关系。交互图包括顺序图和合作图。

顺序图显示对象之间的动态合作关系,它强调对象之间消息发送的顺序,同时显示对象之间的交互。

合作图描述对象间的协作关系,合作图跟顺序图相似,均显示对象间的动态合作关系。除显示信息交换外,合作图还显示对象以及它们之间的关系。如果强调时间和顺序,则使用顺序图;如果强调上下级关系,则选择合作图。

5. 实现图

实现图描述具体实现和实施过程中的部署安排。实现图包括组件图和配置图。

组件图描述代码组件的物理结构及各个组件之间的依赖关系。一个组件可能是一个资

源代码组件、一个二进制组件或一个可执行组件。它包含逻辑类或实现类的有关信息。组件图有助于分析和理解组件之间的相互影响程度。

配置图定义系统中软硬件的物理体系结构。它可以显示实际的计算机和设备(用节点表示)以及它们之间的连接关系,也可显示连接的类型及部件之间的依赖性。在节点内部,放置可执行部件和对象以显示节点与可执行软件单元的对应关系。

从应用的角度看,当采用面向对象技术设计系统时,第一步是描述需求;第二步是根据需求建立系统的静态模型,以构造系统的结构;第三步是描述系统的行为。其中在第一步与第二步中所建立的模型都是静态的,包括用例图、类图(包含包)、对象图、组件图和配置图5个图形,是标准建模语言UML的静态建模机制。第三步中所建立的模型或者可以执行,或者表示执行时的时序状态或交互关系。它包括状态图、活动图、顺序图和合作图4个图形,是标准建模语言UML的动态建模机制。因此,标准建模语言UML的主要内容也可以归纳为静态建模机制和动态建模机制两大类。2.5节将以ATM系统为例,分析ATM系统的用例图,并对“密码修改”用例进行动态建模分析,设计其顺序图,有关其他UML建模语言的实例分析请查阅其他相关文献。

2.5 ATM 系统

ATM是Automatic Teller Machine的缩写,意为自动柜员机,因大部分用于取款,又称自动取款机。它是一种高度精密的机电一体化装置,利用磁性代码卡或智能卡实现金融交易的自助服务,代替银行柜面人员的工作。ATM可完成提取现金、查询存款余额、进行账户之间资金划拨和余额查询等工作。持卡人可以使用信用卡或储蓄卡,根据密码办理自动取款、查询等业务。ATM业务可简单划分为查询余额、取款、存款和更改密码4项功能。

2.5.1 ATM系统分析

ATM系统是一个复杂的软件控制硬件的系统,各功能模块协调工作。首先需要了解整个设备如何协调工作,提供ATM自动取款,存款等服务。ATM系统包含以下的基础模块(见图2.1)。

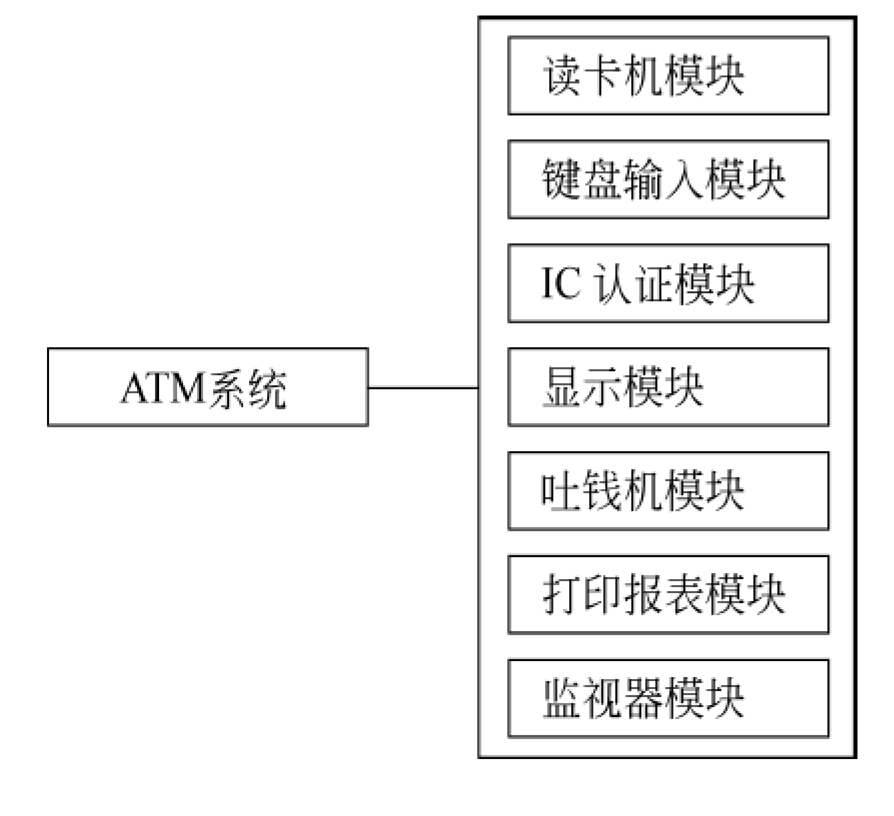


图2.1 ATM系统模块

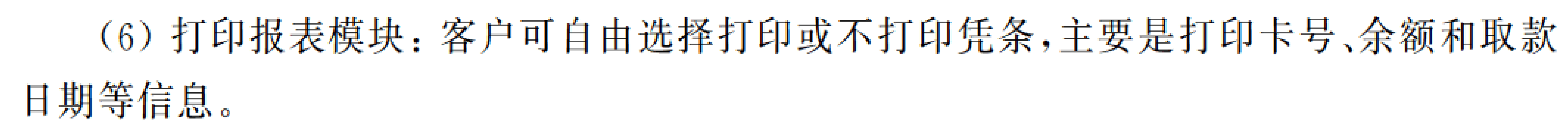
(1) 读卡机模块:客户银行卡插入读卡机,读卡机识别卡,在显示器提示输入密码。

(2) 键盘输入模块:客户通过键盘输入密码与取款金额,并选择要办理的业务。在这个模块中,系统需要和客户进行交互操作。

(3) IC认证模块:基于安全性,鉴别卡的真伪。

(4) 显示模块:显示一切与客户有关的信息,包括交互提示、确认等信息。

(5) 吐钱机模块:根据客户需求,选择合适面值的钞票给客户,这也是关键的一个模块。



(6) 打印报表模块:客户可自由选择打印或不打印凭条,主要是打印卡号、余额和取款日期等信息。

(7) 监视器模块：设置摄像头以防意外事件，保证交易安全。

2.5.2 UML 建模

本节描述使用 UML 语言对 ATM 系统进行建模。

1. 用例图

用例图是由用例、参与者以及用例之间的关系所组成的。

参与者是系统外部的一个实体，可以是人或者事物，它可以参与到用例的执行过程中，形成一些交互的操作，比如系统输入等。ATM 参与者为：①客户，用 ATM 进行现金交易；②银行官员，更改 ATM 设置，放置现金，维护机器；③信用系统，作为外部角色参与整个交易过程。

用例是一个叙述型的文档，用来描述参与者使用系统完成某个事件。识别用例可以从参与者开始，考虑每个参与者如何使用系统以及参与者希望系统提供的功能。

ATM 的用例分析如下：

✍ 客户取款
✍ 客户存款
✍ 客户查询余额
✍ 客户转账
✍ 客户更改密码
✍ 客户通过信用系统付款
✍ 银行官员修改密码
✍ 银行官员为 ATM 添加现金
✍ 银行官员维护 ATM 硬件
✍ 信用系统启动付款功能

用例之间的关系有泛化关系、包含关系和扩展关系。

(1) 泛化关系：用例的泛化关系与类之间的泛化关系相似。子用例和父用例相似，但表现出更特别的行为；子用例将继承父用例的所有结构、行为和关系。子用例可以使用父用例的一段行为，也可以重载它。父用例通常是抽象的。在实际应用中很少使用泛化关系，子用例中的特殊行为都可以作为父用例中的备选流存在。例如，业务中可能存在许多需要部门领导审批的事情，但是领导审批的流程是很相似的，这时可以用泛化关系表示。

(2) 包含关系：把几个用例的公共步骤分离成为一个单独的包含用例。使用包含用例来封装一组跨越多个用例的相似动作，以便多个基用例复用。基用例控制与包含用例的关系，以及包含用例的事件流是否会插入到基用例的事件流中。基用例可以依赖包含用例执行的结果，但是双方都不能访问对方的属性。

包含关系最典型的应用就是复用，也就是定义中说的情景。但是有时当某用例的事件流过于复杂时，为了简化用例的描述，也可以把某一段事件流抽象成为一个被包含的用例；相反，用例划分太细时，也可以抽象出一个基用例，来包含这些细颗粒的用例。这种情况类似于在过程设计语言中将程序的某一段算法封装成一个子过程，然后再从主程序中调用这一子过程。

例如，业务中总是存在着信息维护的功能，如果将它作为一个用例，那么新建、编辑以及修改都要在用例详述中描述，过于复杂；如果分成新建用例、编辑用例和删除用例，则划分太

细。这时包含关系可以用来理清关系。

(3) 扩展关系：是把新行为插入到已有用例的方法。基础用例也可以称为基用例，提供了一些扩展点，在扩展点处可以插入新的行为；而扩展用例提供了一组插入片段，这些片段被插入到基用例的扩展点处。将基用例中一段相对独立并且可选的动作用扩展用例加以封装，再让它从基用例中声明的扩展点上进行扩展，从而使基用例行为更简练，目标更集中。扩展用例为基用例添加新的行为。扩展用例可以访问基用例的属性，因此它能根据基用例中扩展点的当前状态来判断是否执行自己。但是扩展用例对基用例不可见。对于一个扩展用例，可以在基用例上有几个扩展点。

例如，系统中允许用户对查询的结果进行导出和打印。对于查询而言，无论能不能导出和打印，查询都是一样的，导出和打印是不可见的。导出、打印和查询相对独立，而且为查询添加了新行为。因此可以采用扩展关系来描述。

ATM 的用例图表示在参与者与用例之间存在交互关系，客户、银行官员和信用系统这 3 个参与者都有各自的用例关系图。综合所有参与者的用例关系，可以建立整个 ATM 系统的用例关系图，如图 2.2 所示。

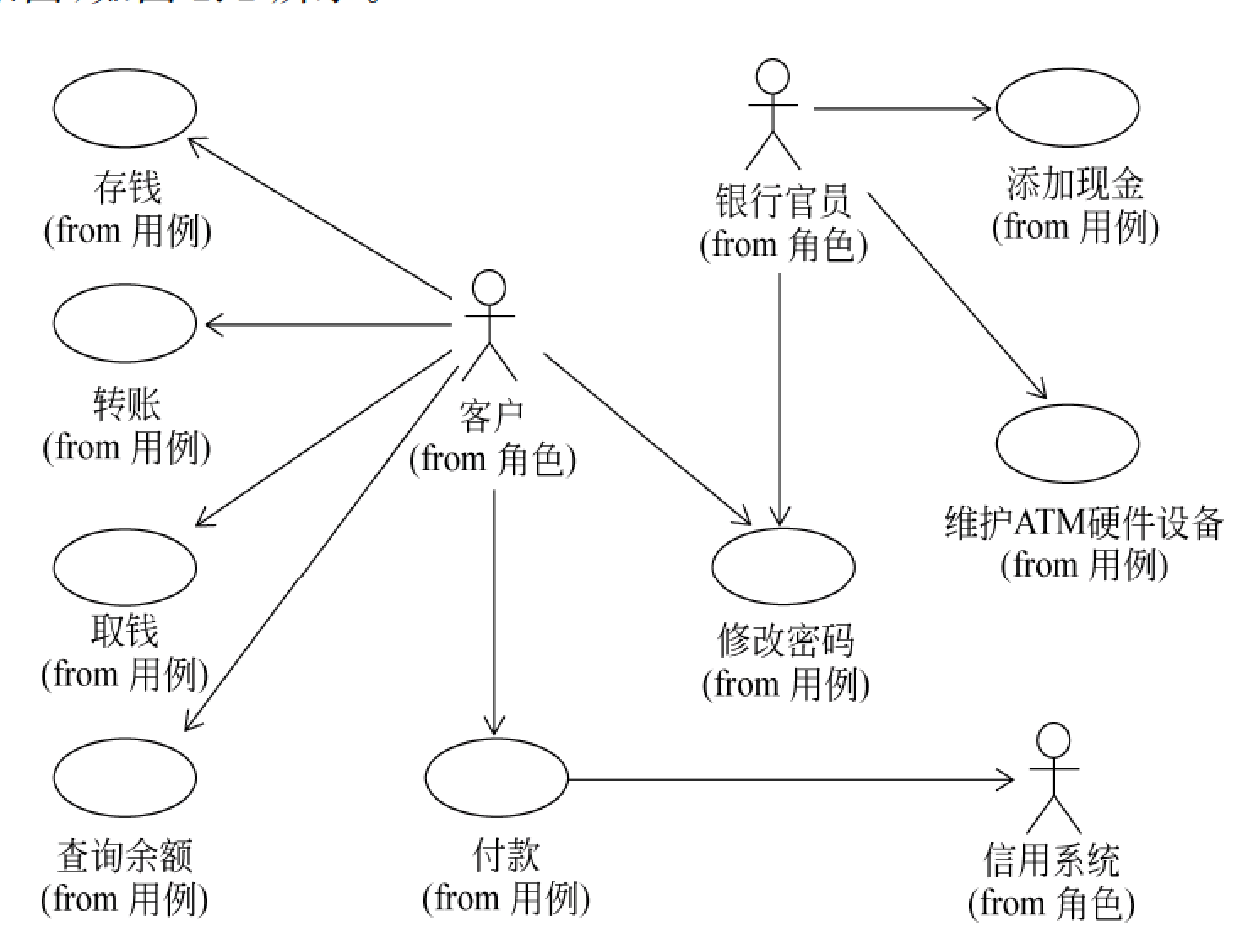


图 2.2　ATM 系统用例关系图

模型中的活动者代表外部与系统交互的单元，包括客户、银行官员和信用系统；业务用例框图是对系统需求的描述，表达了系统的功能和所提供的服务，包括客户现金交易子系统、银行管理维护子系统、客户服务子系统和信用子系统。

2. 交互图

交互图用来表达对象之间的交互关系，分为两种：顺序图和合作图。

顺序图强调消息发送的时间顺序，合作图则强调接收和发送消息的对象的组织结构。两者在语义上是等价的，可等价转换，更改其中一个框图时，另一个框图会发生同步变化。

顺序图的功能是按照时间和空间顺序描述系统元素间的交互和它们之间的关系。顺序图中包括的建模元素主要有类角色、生命线、激活期和消息等。

(1) 类角色：表示顺序图中对象在交互中所扮演的角色，类角色一般代表实际的对象，比如 ATM 系统中的某客户：Susan。

(2) 生命线：在顺序图中表示为从对象图标向下延伸的一条虚线，表示对象存在的时间。

(3) 激活期：代表对象执行一项操作的时期，表示在这个时间段内对象将执行相应的操作，用小矩形表示。

(4) 消息：是定义交互和协作中交换信息的类，用于对实体间的通信内容进行建模。消息一般分为同步消息(synchronous message)，异步消息(asynchronous message)和返回消息(return message)。

同步消息：消息的发送者把控制传递给消息的接收者，然后停止活动，等待消息的接收者放弃或者返回控制。用来表示同步的意义。

异步消息：消息发送者通过消息把信号传递给消息的接收者，然后继续自己的活动，不等待接收者返回消息或者控制。异步消息的接收者和发送者是并发工作的。

返回消息：表示从过程调用返回。

顺序图中还常用一种消息，称为自关联消息(self-message)，它表示方法的自身调用以及一个对象内的一个方法调用另外一个方法。

这里对部分用例事件流程进行建模。如图 2.3 所示，以客户 susan 修改密码为例，显示

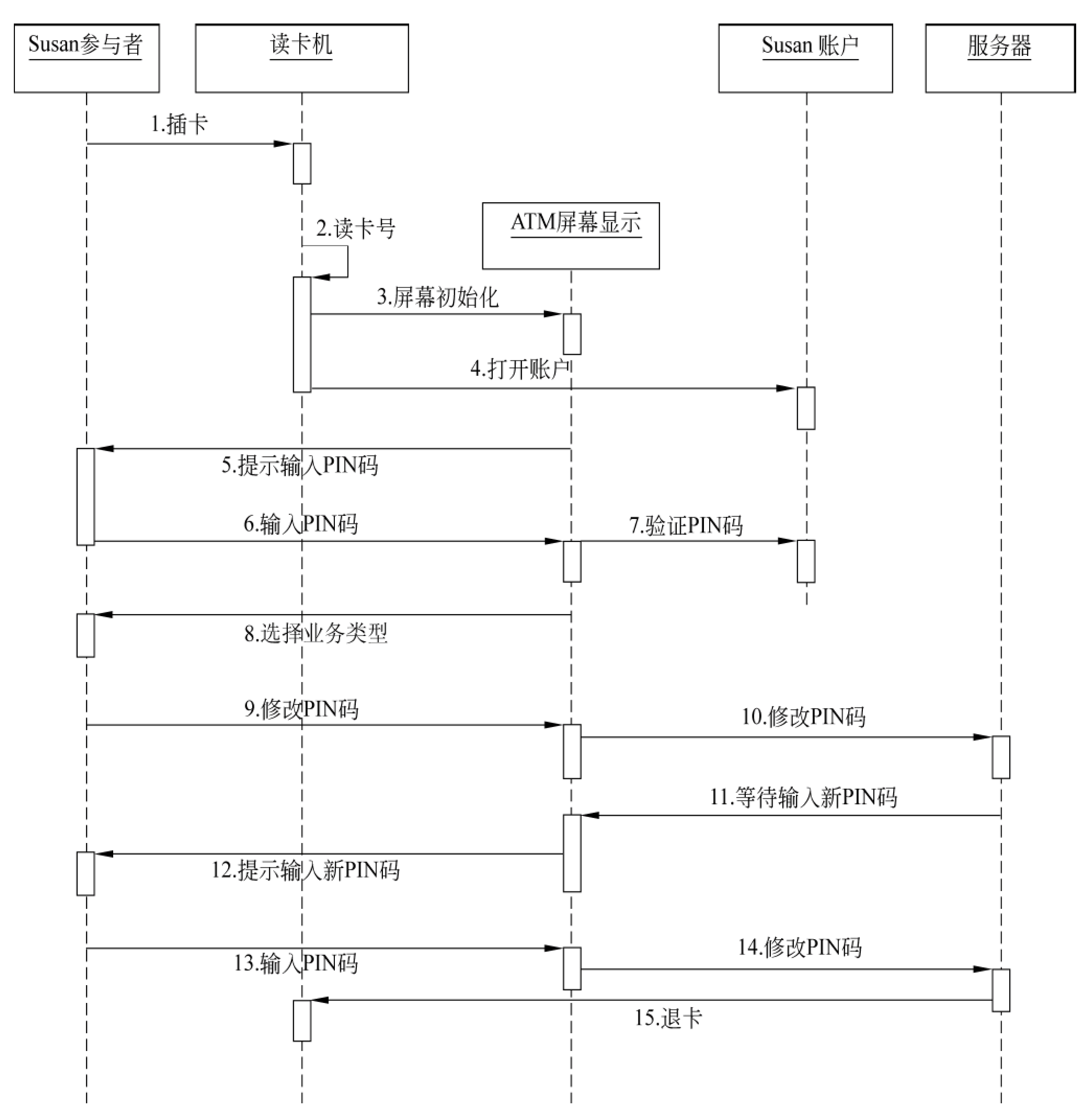


图 2.3 “修改密码”顺序图

“修改密码”用例中的功能流程。顺序图中用箭头表示在各个对象间传递消息，这些箭头要么指向其他对象，要么自反。箭头的方向表示信息的流向，可以流向其他对象，也可以流向对象本身。步骤用数字来标志。

Susan 如果需要修改密码，首先将磁卡插入 ATM 系统，等待 ATM 系统读卡机工作，验证卡片。如果验证成功，系统会进入 Susan 个人账户，并等待 Susan 输入 PIN 码与 Susan 账户中的 PIN 码比对，进行 PIN 码验证功能。PIN 码验证成功后，ATM 屏幕会显示出可供选择的业务供 Susan 选择。由于图 2.3 只是针对密码修改进行时序分析，Susan 会选择修改密码功能，录入新的密码。

2.6 本章小结

本章作为后续章节学习的基础，主要介绍了两个程序示例，分别是逻辑结构比较复杂的 NextDate 函数和 ATM 系统。本章首先介绍了通用伪代码和伪代码的语法规则，接着对 NextDate 问题进行分析、讨论并进行最简单的实现；在介绍 ATM 系统之前，首先介绍了面向对象方法——统一建模语言(UML)的相关的概念，最后采用 UML 的形式来描述 ATM 系统。

习 题

1. 阅读下面的伪代码，确定该伪代码的含义。

```
Program average grade
Dim grade,sum,average As Real
Dim counter As integer
sum=0.0
counter=0
Do While there no more data
      input(grade)
      sum=sum+grade
      counter=counter+1
EndWhile
average=sum/counter
Output(average)
End average grade
```

2. 三角形问题。接收 3 个整数 a、b 和 c 作为输入，用作三角形的边。程序的输出是由这 3 条边确定的三角形类型：等边三角形、等腰三角形、不等边三角形或非三角形。其中，整数 a、b 和 c 必须满足以下条件：

c1：$1\leqslant a\leqslant 200$　　c4：$a<b+c$

c2：$1\leqslant b\leqslant 200$　　c5：$b<a+c$

c3：$1\leqslant c\leqslant 200$　　c6：$c<a+b$

要求写出该问题的伪代码并进行实现。

3. 参考本章中的 NextDate 问题示例,写出伪代码。

4. 在 UML 提供的图中,________用于描述系统与外部系统及用户之间的交互;________用于按时间顺序描述对象间的交互。

5. 画出下面"借书"场景的顺序图。

(1) 管理员启动一次借书(makeNewCase())。

(2) 管理员输入图书标识(enterItem(itemID))。

(3) 管理员输入借阅者信息(enterReader(ReaderID)),系统计算并显示借阅者、借书日期和还书日期等信息(endCase())。

(4) 图书消磁,借阅者获得相应书籍(makePayment(amount))。

第 3 章　软件测试用例的设计

在软件测试过程中,测试用例的设计是软件测试的灵魂。测试工程师就是借助测试用例的运行来检测被测软件的功能和性能。测试结果的广度和深度完全由测试用例来决定。完全覆盖测试是最容易想到的一类用例,它要求测试工作的力度和深度以及每一种现实中可能发生的操作都保证正确无误。软件测试中永远不可能做到穷举测试,然而又想使测试工作的效率达到最高,那么,如何兼顾工作量和工作效率往往成为测试工作中的瓶颈问题。如何测试,用什么方式来测试,在什么环境和什么样的条件下进行测试,如何减少测试的工作量和如何避免重复的测试等,测试用例都应该考虑在内。在测试工作中如何协调和同步,在测试用例也应该充分描述这些问题。

经典软件测试理论有很多测试用例设计方法,总的来说,这些方法可以分为两大类,一类是黑盒测试,一类是白盒测试。

随着面向对象技术的发展,尤其是 UML 语言的广泛应用,测试用例设计又遇到了许多新的问题。为了适应面向对象这一新的技术,人们提出了很多实用的面向对象测试用例设计方法。例如,为了解决面向对象技术中大量的消息传递,对于对象交互等方面的测试,可以借助 MM(Method/Message)图来进行测试用例设计。MM 图是一种新的路径测试方法,与普通的 DD(Decision-to-Decision path)路径图比较,它更凸显了对象的交互。

本章将系统的介绍黑盒、白盒及面向对象测试用例设计的基本内容,下节将会详细的讲解黑盒测试及其测试用例设计方法。

3.1　黑 盒 测 试

黑盒测试是一种常用的软件测试方法。据统计,将近 80%的软件缺陷都是利用黑盒测试完成的。那么什么是黑盒测试呢?黑盒测试强调了软件输入与输出之间的关系,它将被测软件看作一个打不开的黑盒,根据软件规格说明书设计测试用例,完成测试。

我们学习过“函数”的定义,函数是把一个集合(定义域)的值映射到另一个集合(值域)上面。从某种意义上,可以把软件(程序)的输入和输出看作是函数的定义域和值域。黑盒测试就是从这种观点出发,测试人员把软件(程序)看作一个打不开的黑盒,只关心输入与输出的结果。

本节主要介绍几种常用的黑盒测试方法,并通过实例介绍各种方法的运用。之后,在第 7 章将介绍几种常用的黑盒测试工具,并结合实例重点介绍 IBM Rational 系列的 Function Tester 测试工具的应用。

3.1.1　等价类测试

不论使用哪种测试用例设计方法,其根本意义都是要减少测试用例数量。然而在减少测试用例数量的同时,还希望测试工作找出更多的错误。因此测试工程师就想到借助于“等

价类”的思想来建立测试用例集，使得在减少测试用例绝对数量的同时能够保证测试用例的质量。

首先讨论等价类的基本概念以及它在测试过程中的重要意义。在分析等价类之前，需要了解等价关系的概念。

等价关系 R 为非空集合 A 上的关系，如果 R 是自反的、对称的和传递的，则称 R 为 A 上的等价关系。那么什么是等价类呢？假设 x 为 A 上的元素，那么 x 的等价类记为 $[x]_R$，并满足：$[x]_R=\{y|y\in A\wedge xRy\}$，那么等价类中的元素均满足等价关系。

显然，如果将集合 A 看作针对某一个功能所设计的测试用例集合，那么 x 等价类内的元素都是等价的。也就是说，理论上只取 $[x]_R$ 中某一个测试用例来进行测试，和取 $[x]_R$ 中每一个测试用例进行测试所得到的测试结果应该是相同的。这样，利用等价类的概念可以找到冗余的测试用例，用以减少测试用例的绝对数量。

那么如何来进行等价类划分呢，在进行等价类划分时一定要注意两点，一个是划分的完备性，另一个是冗余性。一般情况下，在进行测试的过程中，使用等价类划分方法，都是针对程序的输入输出域来进行等价类划分，从而设计等价类和相应的测试用例。

1. 等价类划分方法

在了解等价类划分的定义之后，先来看一个等价类划分的小例子。

例 3.1 图 3.1 是一个简单的 login 界面，在绝大部分 MIS 系统中，都需要用到的 login 这一简单又很重要的模块。



图 3.1 登录界面

图 3.1 为学生信息管理系统的登录界面，在规格说明书中对登录模块的描述如下。

要求 1：用户名使用学生学号，学号要求由 9 位数字组成，如 090705101。

要求 2：密码使用 4～8 位字符串。字符串由大小写字母、下划线“_”或数字组成。

很明显，该例子中使用等价类划分，可以针对输入域进行等价类分析。

分析：

(1) 要求用户名(也就是学号)由 9 位数字组成，等价类划分如下。

等价类 1：9 位数字，为一组等价类(090705101；070405206；100705102)；

等价类 2：非 9 位数字，为一组等价类(0907051；10070)；

等价类 3：用户名中含有字母和其他字符。

(2) 要求密码使用 4～8 位字符串，等价类划分如下。

等价类 4：4～8 位字符串，为一组等价类；

等价类 5：非 4～8 位字符串，为一组等价类。

(3) 要求字符串由大小写字母、下划线“_”或数字组成，等价类划分如下。

等价类 6：字符串包含大小写字母、下划线“_”或数字；

等价类 7：字符串包含特殊字符(空格、￥、＃、@等)。

从例 3.1 可以看出，等价类划分方法就是将输入域(输出域)进行等价划分。划分过程一般是按照规格说明书的内容，由测试工程师根据实际情况来操作的。也就是说，不同的测

试人员设计出的等价类不一定是相同的。因此，在具体操作的时候，整个等价类设计过程要遵循这样一个原则：在划分等价类时要考虑到无效和未预料到的情况，这一点在输出域等价类划分过程中尤其重要。根据这一原则，把等价类划分为有效等价类和无效等价类。

(1) 有效等价类：符合程序规格说明书，有意义的、合理的输入(输出)数据所构成的集合。

(2) 无效等价类：不符合程序规格说明书，不合理的或者无意义的输入(输出)数据所构成的集合。

一般在具体的问题中，有效等价类可以是一个，也可以是多个；而无效等价类至少应有一个。对于例3.1，很明显，等价类1、4、6为有效等价类，等价类2、3、5、7为无效等价类。

另外，在等价类划分的时候一定要注意划分的完备性和非冗余性。完备的划分保证了测试用例能够覆盖所有的输入域，没有遗漏；非冗余性使得划分更加合理，测试用例质量更高。

2. 等价类划分原则

虽然等价类划分是一种随机性比较强的测试方法，不同测试人员会得到不同的划分结果，但是还是有一定规律可循的。这些规律是很多测试人员在工作中总结提炼而得到的，有一定的通用性，这对初学者而言是很有帮助的。

(1) 区间划分：规格说明对数据规定了明显的取值范围。比如，血压值为50～200，学生学号为070000001～120000001。对这一类数据，等价类可取区间内(有效等价类)数值和区间外(无效等价类)数值。

(2) 集合划分：规格说明对数据规定了明确的集合，比如管理员(admin和superadmin)。等价类可取集合内(有效等价类admin)和集合外(无效等价类Tom)不同集合值。

(3) 特殊规则划分：规格说明规定了数据必须遵守一定的限制条件。比如，根据例3.1中的要求2，可以确定等价类4满足规则说明，等价类5不满足规则说明。

(4) 细分等价类：等价类在具体程序处理时发现并不真正等价。比如，学号090705101、070405206和100705102分别为该校2009级学生、2007级学生和2010级学生，那么上述学号在选课、学籍管理等功能模块中并不等价。此时可以根据具体情况来细分相应的等价类。

3. 等价类划分测试用例设计

在进行测试用例的具体设计时，也要考虑到有效等价类和无效等价类。一般情况下，我们希望一个测试用例能够覆盖较多的有效等价类；同时，一个测试用例覆盖并且只能覆盖一个无效等价类，这是由有效等价类和无效等价类的特点决定的，其目的是防止第一个无效等价类的测试会屏蔽或者终止其他无效等价类的测试执行。比如，例3.1设计的测试用例如表3.1所示。

表3.1 测试用例

测试用例	输 入	期望输出	覆盖等价类
T1	用户名：090705101 密码：a_bcde	满足登录条件数据库查询匹配情况	1,4,6
T2	用户名：0907051 密码：a_bcde	提示：用户名为9位的学号	2
T3	用户名：0907051a 密码：a_bcde	提示：用户名为9位数字串	3

续表

测试用例	输　　入	期 望 输 出	覆盖等价类
T4	用户名：0907051a 密码：a_b	提示：密码为4～8位字符串	5
T5	用户名：0907051a 密码：a@b	提示：密码为4～8位字符串，不能包含特殊字符	7

4. 等价类划分方法小结

一般进行等价类划分需要两个步骤：

(1) 根据规格说明书的内容对输入域(输出域)划分等价类。

(2) 根据划分的等价类进行测试用例设计。

具体等价类执行步骤如下：

(1) 划分等价类，将每一个等价类进行编号。

(2) 标记等价类是有效等价类还是无效等价类。

(3) 设计测试用例。

① 设计有效等价类测试用例，尽可能多地覆盖所有有效等价类。

② 设计无效等价类测试用例，一个无效等价类测试用例只能覆盖一个无效等价类。

利用等价类法测试，结果如表3.2所示。读者可按照表3.2的形式设计测试用例，以防遗漏某一等价类式测试条件。

表3.2　等价类表

输入(输出)条件	编号	有效等价类	无效等价类

等价类划分经常和边界值分析联合起来使用。如果等价类划分的数据是独立的物理数据，也就是满足边界值测试条件的话(比如“血压(50～200)”)，可以取等价类的边界值(50，60，75，190，200)。

3.1.2　边界值测试

大量的软件测试实践表明，软件缺陷经常出现在物理数值的边界上，因此利用边界值分析方法进行软件测试，是一种很实用的方法，它具有很强的故障检测能力。这一测试方法适用于分析独立的变量，而这一变量常常是物理量(如温度、速度等)。边界值测试方法所分析的物理量，可以从两方面来考虑，一方面是针对软件输入输出中所使用的数据，另一方面是程序内部所涉及到的一些关键性的物理数据。

举一个简单的例子，程序员由于疏忽，在进行程序设计时，将图3.2所示的程序段，写成图3.3所示的程序段，造成很难发现的软件缺陷。

图3.2和图3.3提供两个程序段，类guessNumb为猜数游戏类。图3.2可以正确地得到用户想要的结果。图3.3为存在缺陷的程序段，但是如果不使用边界值“39”很难发现该缺陷。所以边界值是一个很实用的，并且是不可或缺的测试用例设计方法。

边界值分析方法的基本思想就是利用输入变量的边界处的值进行测试。一般情况下，

```
import java.util.Scanner;
public class guessNumb{
  public static void main(String args[]){
    int y=39;
    Scanner reader=new Scanner(System.in);
    System.out.println("输入您猜测的数字（0-100）")
    int x=reader.nextInt();
    If(x<y){
      System.out.println("您猜测的数字偏小")；
    }
    Else{
      If(x==y)
        System.out.println("太聪明了")
      Else
        System.out.println("您猜测的数字偏小")
    }
  }
}
```

图 3.2　程序段 1

```
import java.util.Scanner;
public class guessNumb{
  public static void main(String args[]){
    int y=39;
    Scanner reader=new Scanner(System.in);
    System.out.println("输入你猜测的数字（0--100）")
    int x=reader.nextInt();
    If(x<=y){
      System.out.println("您猜测的数字偏小")；
    }
    Else{
      If(x==y)
        System.out.println("太聪明了")
      Else
        System.out.println("您猜测的数字偏小")
    }
  }
}
```

图 3.3　程序段 2

所谓边界值包括最小值、最大值、略大于最小值的值、略小于最大值的值以及正常值。

1. 边界值测试方法

1）单变量边界值分析

边界值分析法主要着眼于输入空间的边界。其中最常用的方法莫过于"五点法"，也就是上面所说的取变量的最小值、最大值、略大于最小值的值、略小于最大值的值以及正常值这 5 个值。

例 3.2　分析整型变量 x(a≤x≤b)，区间[a,b]为变量 x 的取值范围。

分析：利用边界值分析方法进行测试用例设计。

最小值：a(记为 min)；

略大于最小值的值：a＋1(min＋)；

正常值：c(a+1≤x≤b−1)(nom)；

略小于最大值的值：b−1(max−)；

最大值：b(max)。

利用以上5个值来进行五点法边界值测试。

此例是针对单一变量x进行边界值分析，显然用五点法就可以。但是针对物理量的不同，在选取“略大于最小值的值”和“略小于最大值的值”时要考虑步长的问题。

例3.3 某个输入文件最多可容纳255条记录，根据五点法进行边界值分析。

分析：利用边界值分析方法进行测试用例设计。

(1) 将文件放入1条记录，测试应用程序(min)；

(2) 将文件放入2条记录，测试应用程序(min+)；

(3) 将文件放入50条记录，测试应用程序(nom)；

(4) 将文件放入254条记录，测试应用程序(max−)；

(5) 将文件放入255条记录，测试应用程序(max)。

例3.4 某企业员工月工资从1000到2000不等，个人所得税费按月扣除，计算得最小金额从0.00元到最大金额4646.89元不等，根据五点法进行边界值分析。

分析：利用边界值分析方法进行测试用例设计。

(1) 测试扣除个人所得税0.00(min)；

(2) 测试扣除个人所得税0.01(min+)；

(3) 测试扣除个人所得税2000(nom)；

(4) 测试扣除个人所得税4646.88(max−)；

(5) 测试扣除个人所得税4646.89(max)。

例3.3和例3.4同样是采用五点法进行边界值分析，在例3.3中所采用的步长为1，而例3.4中所采用的步长为0.01。步长的选择要根据实际情况来分析。

2) 多变量边界值分析

以上分析的问题都是针对于一个变量的边界值用例设计，而作为程序的输入(输出)往往都是多个变量同时参与计算过程中。那么当变量数大于1时，如何采用五点法进行边界值分析呢？

下面讨论两个变量x、y，并且给出x和y的边界条件(取值范围)：

$$a\leqslant x\leqslant b;c\leqslant y\leqslant d$$

根据五点法的要求，选取(x_{min}, y_{nom})，(x_{min+}, y_{nom})，(x_{nom}, y_{nom})，(x_{max-}, y_{nom})，(x_{max}, y_{nom})，(x_{nom}, y_{min})，(x_{nom}, y_{min+})，(x_{nom}, y_{max-})，(x_{nom}, y_{max})。如图3.4所示。同样的道理，对于n个变量的程序，采用五点法将产生$4n+1$个测试用例。

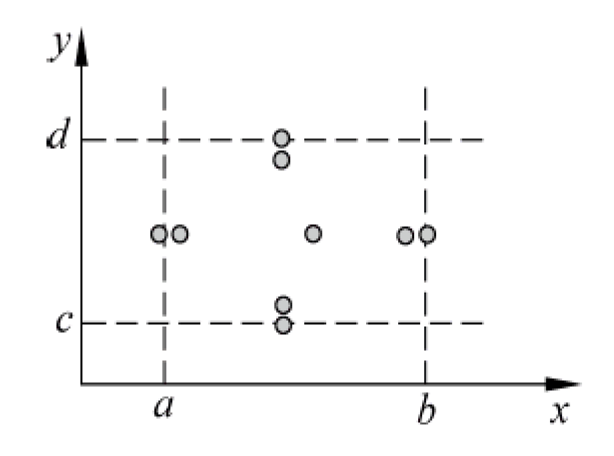


图3.4 多变量五点法测试用例设计

3) 健壮性边界值分析

“健壮性”这个词，经常出现在软件测试领域，包括系统测试时的健壮性测试和这里的健壮性边界值分析。有关健壮性的测试往往是检测无效的未预料到的输入和输出。尤其在无效的输出方面，健壮性测试有着不可小觑的能力。

健壮性边界值分析也常常被称为“七点法”边界值测试。也就是对于单变量x(a≤x≤b)

而言，除了考虑五点法中的五点：min、min+，nom，max－和max之外，还需要分析略小于最小值的值min－和略大于最大值的值(max+)。同样的道理，对于多变量健壮性边界值分析时，假设变量数n，需要设计$6n+1$个测试用例。

2. 软件输入输出中存在的边界问题

在进行软件输入输出边界测试时，需要注意以下一些方面。

(1) 测试的数据类型，其中包括数值、字符、位置、数量、速度、地址和尺寸等。

(2) 测试数据的边界特征值，如第一个/最后一个、开始/完成、空/满、最慢/最快、相邻/最远、最小值/最大值、超过/在内、最短/最长、最早/最迟、最高/最低等。此外还包括以下几项：

① 一些特殊的字符：@、'、{、/、:等，以及特殊字符的ASCII码。

② 常用的边界还包括：数字0～9，对应ASCII码48～57；大写字母A～Z，对应ASCII码65～90；小写字母a～z，对应97～122。

③ 对于图形设计类程序，还需考虑显示范围的边界、光标最上端和光标最下端的边界等内容。

3. 程序内部所涉及的边界问题

在图3.3所示的程序段中，软件缺陷并非是输入输出变量导致的，也就是说，在进行软件测试时，还需要考虑程序内部某些关键变量的边界值。

在进行程序内部变量边界值测试时，需要注意以下几个方面的内容。

(1) 计算机和软件的基础是二进制数。因此与2的乘方相关的值是作为边界条件的重要数据。

例如，在通信软件中，带宽或者传输信息的能力总是受限制的，因此软件工程师会尽一切努力在通信字符串中压缩更多数据。其中一个方法就是把信息压缩到尽可能小的单元中，发送这些小单元中最常用的信息，在必要时再扩展为大一些的单元。假设某种通信协议支持256条命令。软件将发送编码为一个双位数据的最常用的15条命令；如果用到第16～256条命令，软件就转而发送编码为更长字节的命令。这样，软件就会根据双位/字节边界执行专门的计算和不同的操作。

(2) 对于数组型数据int a[]=new int[20]，以a[0]以及a[19]作为其边界值。

(3) 对于循环语句

```
for(i=1;i<=n;i++){
循环体
}
```

第0次循环和第n次循环作为其边界条件。

(4) 对堆栈类型数据结构的数值进行测试时，对于该堆栈而言，为空(null)、满(full)分别可作为其边界条件。

当然，还有其他很多种涉及数值型、字符型、速度值以及其他物理量的数据，都可以采用边界值方法对其进行测试。

4. 边界值法测试用例设计的局限性

边界值分析方法具有很强的缺陷发现能力，是一种设计简单，但很实用的测试方法。边界值设计测试用例方法适合于分析独立的变量，并且这些变量表示的是实际的物理量(如速

度、时间、温度等)。也就是说,边界值分析方法所测试的变量要求是独立的并且是物理量。而边界值测试方法就是取这些物理量的临界值或极值作为测试数据。

例如,由于1992年6月26日菲尼克斯气温达到122℉,菲尼克斯的天港国际机场被迫关闭,而此次事件仅仅因为机场部分设备接收的气温上限是120℉。再举个例子,在飞机飞行过程中,如果飞机的飞行仰角达到其极限,很可能会造成严重的飞行事故。也就是说,在应用程序中,对于含有实际物理意义的变量进行边界值分析是非常重要的,对于某些变量来说也是必不可少的一个过程。

边界值分析方法并不是不能分析有依赖关系的变量,比如经典的三角形问题中(a,b,c为输入的3条边),依然可以采用边界值方法进行分析,但是由于边界值方法的特点,这种设计用例的过程并不能考虑到变量之间的关系,比如a<b+c这样的依赖关系。所以一般认为,使用边界值分析方法分析多变量时,多变量之间的关系是独立的,这也正是边界值分析方法存在的很大的缺陷。

3.1.3 因果图

等价类划分和边界值分析是很有效的测试方法,但是当多变量之间关系不是独立的,有复杂的逻辑关系的时候,这两种方法就显得束手无策了。举个简单的例子,某传感器误差Δx随着温度的影响累计增大,可表示为$\Delta x=f(\Delta c)$,其中Δc为温度的变化,当Δx超过阈值d时发生失效,那么等价类和边界值方法在处理这些问题时,很难发现这种类型的失效。为了很好地表示出输入数据之间的组合逻辑关系,可以借用"因果图"这一有效的方法。

因果图法是由美国IBM公司的Elemendorf在吸收了硬件测试中自动生成逻辑组合电路测试等技术的基础上于1973年提出的,它是进行功能测试时把功能说明书形式化的一种表示方法。因果图是一套严谨的知识表示方式,它类似于数字逻辑电路,可以清晰地表达组合数据之间的布尔逻辑关系。使用因果图方法表示数据,不但有助于测试工作的进行,还可以发现规格说明中描述不完整或者存在二义性的内容。一般在使用因果图来表示输入域或输出域时,往往还需借助决策表来进行测试用例设计。关于决策表的内容在3.1.4节将会作详细的介绍。

1. 因果图中使用的符号

由于因果图表示的数据主要体现了数据之间的布尔逻辑关系,所以可借助数字逻辑电路的与(∧)、或(∨)、非(~)等符号表示。在因果图中,用圆圈表示节点(原因或者结果),用直线连接相应的节点。

在连接因果的直线上面可以标出节点之间的关系。如果是原因节点和结果节点,那么节点之间的关系主要有以下4种:

(1) 恒等:如果原因为真,结果必为真。

(2) 非:如果原因为真,结果必为假。

(3) 或:如果原因组合(如$c_1 \vee c_2 \vee c_3$)为真,结果为真。

(4) 与:如果原因组合(如$c_1 \wedge c_2$)为真,结果为真。

节点间的上述4种关系用因果图来表示如图3.5所示。

因果图除了明确给出原因和结果之间的关系,还给出原因之间的约束关系。原因之间的约束关系有以下4类。

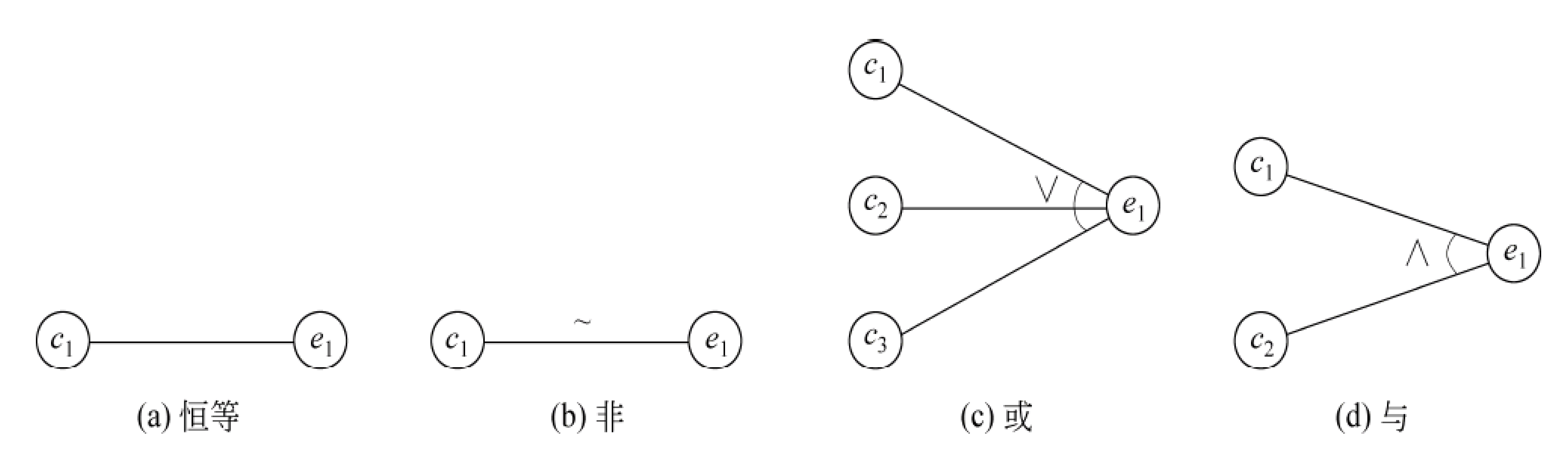


图 3.5 因果图中原因和结果之间的关系

(1) Exclusive(E)：原因 a、b 不可能同时设置为 1。也就是说 a、b 之间存在排斥的关系。在实际情况中，原因之间的 E 关系比较多，比如，a 字符串首字符为字母 a～z，b 字符串首字符为数字。a、b 两种情况不可能同时成立，存在排斥关系。

(2) Inclusive(I)：原因 a、b、c 至少有一个为 1。a、b、c 不能同时为 0。

(3) Only one(O)：原因 a、b 必须有一个为 1，且仅有一个为 1。

(4) Require(R)：原因 a 是 1 时，要求 b 必须为 1。也就是说，a 为真对 b 有制约关系。

有时结果之间也需要建立约束关系，如果结果 a 为 0，则 b 强制为 0。这种约束关系用 Mask(M)来表示。

因果图中的约束关系如图 3.6 所示。

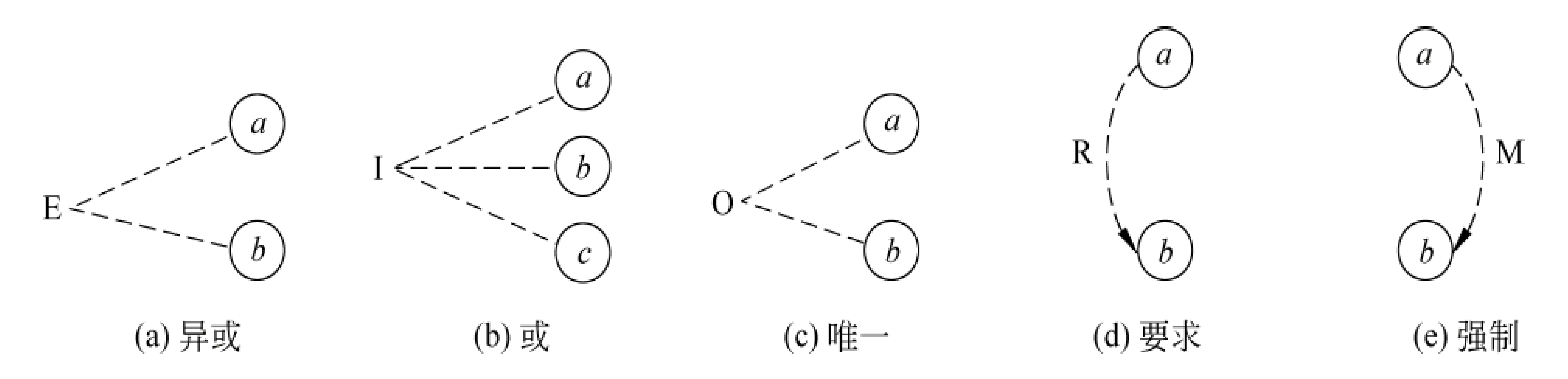


图 3.6 因果图中的约束关系

2. 因果图示例

利用因果图进行测试时，首先要明确测试内容，做到有的放矢。把规格说明中需要测试的内容(比如图书管理系统中的图书信息查询功能)找到，利用因果图对规格说明书的内容进行形式化表示。“因”一般指输入条件或者输入条件的等价类；“果”一般指输出条件，或者后续将要进行的操作，或者系统状态转换等。

例 3.5 在某个嵌入式软件报警系统中，如果设备超速运行，则立即发出消息：“警告：该系统超速运行，误差将会增大！”；如果环境发生变化，则立即发出消息：“错误：系统环境不符合要求，请立即关闭软件！”；如果操作员发生误操作，则立即发出消息：“错误：请重新操作！”

分析：系统故障分为 3 类，即设备超速、环境不符合要求以及操作员误操作。

下面细分这 3 类故障。

(1) 设备超速：可能是由于加速度 $\alpha > a$(固定值)，或者速度 $v > b$(固定值)。

(2) 环境不符合要求：可能是清晰度太差导致软件运行不正常，或者环境的温度或湿度不符合要求。

(3) 操作员发生误操作：操作员操作顺序或者数据输入错误，从而导致系统状态转换错误，或者计算结果发生错误。

首先根据规格说明书确定原因和结果。

原因：c_1：加速度 $\alpha > a$。

c_2：速度 $v > b$。

c_3：清晰度太差。

c_4：温度不符合要求。

c_5：湿度不符合要求。

c_6：操作顺序错误。

c_7：输入数据错误。

c_8：系统状态转换错误。

c_9：计算结果错误。

结果：e_1："警告：该系统超速运行，误差将会增大！"

e_2："错误：系统环境不符合要求，请立即关闭软件！"

e_3："错误：请重新操作！"

然后找到原因和结果之间的因果关系，以及原因和原因之间的约束关系，得到因果图，如图 3.7 所示。

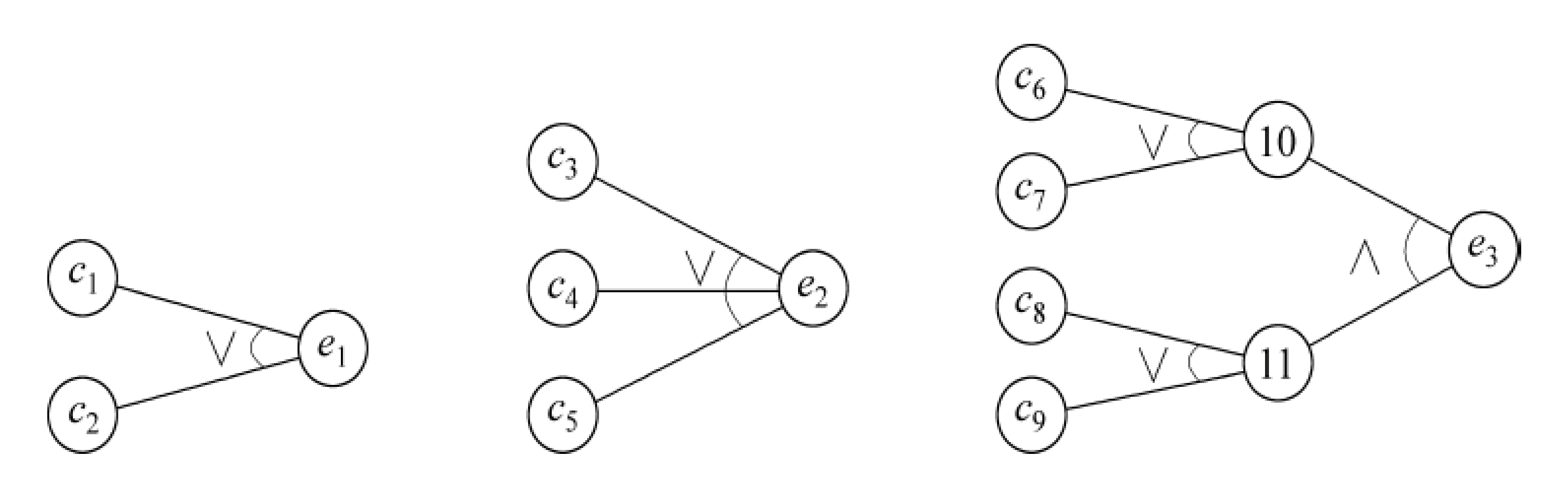


图 3.7 例 3.5 的因果图

以上只是因果图的一个简单的例子，事实上，因果图在处理复杂的问题时是很有效的一个方法，它不但能够有助于完成测试，还能很好地表示功能需求，发现功能需求中存在二义性和不完整的内容。那么接下来的测试用例设计，还需借助决策表或层次树来进行分析。3.1.4 节将介绍借助因果图和决策表对例 3.5 进行测试用例设计的方法。

3.1.4 决策表

决策表是最具逻辑性的测试方法，和因果图法有重叠的地方。决策表可以用来分析和表达多逻辑条件下执行不同操作的程序。自 20 世纪 60 年代以来，决策表一直被用来描述对象之间的复杂逻辑关系。

1. 决策表基本概念

首先介绍决策表的基本内容。一个决策表由 4 部分组成，分别是条件桩、条件项、动作桩和动作项。

(1) 条件桩：列出问题的所有条件。

(2) 条件项：针对条件桩给出的条件列出所有可能的取值。

(3) 动作桩：列出问题规定的可能采取的操作。

(4) 动作项：指出在条件项的各组取值情况下应采取的动作。

另外，一个条件组合的特定取值和相应要执行的操作称为一条规则。在决策表中，一列就是一条规则。

下面通过一个简单的例子学习决策表的应用。

例 3.6 在图书管理系统中，还书模块，管理员录入书目信息。还书过程中利用读者借书的时间以及对书的损坏程度等因素来计算罚款数、读者信誉和读者级别，完成还书操作。

还书操作具体说明如下：

读者级别如表 3.3 所示。

表 3.3 读者级别

读者级别	借阅时间/月	超期罚款额/元/(天·本)	可借本数
SVIP	5	0.2	10
VIP1	2	0.3	5
VIP	1	0.5	3

损坏程度分为 3 个级别，分别为正常磨损、损坏和严重损坏(丢失)。

读者信誉分为 10 个层次，当读者按时还书并且书处于正常磨损状态，读者信誉自动加 1；如果书处于严重损坏或者丢失状态，读者级别自动降为 VIP 级；如果读者超期时间超过 1 年，读者级别也自动降为 VIP 级别；如果书处于损坏状态，信誉自动减 1。当读者达到最高信誉时，读者级别可以自动提升一级。

分析：该图书管理系统的还书操作计算较复杂，变量之间存在明显的逻辑关系。考虑使用决策表方法进行测试用例分析(这里假设用户级别为 VIP1。读者可自行设计 VIP 和 SVIP 级别的还书操作决策表)。

条件桩：问题的所有条件。

C1：读者借书时间；

C2：对书的损坏程度。

条件项：对条件桩给出的条件列出所有可能的取值。

C1：C 正常(Normal)，E 超期(Exceed)，S 严重超期(Super Exceed)；

C2：C 正常(Normal)，E 损坏(Destory)，S 严重损坏(Super Destory)。

动作桩：出现问题时按规定可能采取的操作。

A1：交罚款；

A2：信誉加 1；

A3：信誉减 1；

A4：降级为 VIP。

动作项：指出在条件项的各组取值情况下应采取的动作。

分析这个过程，建立决策表，根据规格说明填写动作项，并完成决策表的建立。本例的决策表如表 3.4 所示。

决策表适合于描述具有复杂逻辑条件的程序，它能够将复杂的问题按照各种可能的情况全部列举出来，表示形式简洁清晰。还可反过来帮助需求分析人员找到描述不准确或者

表 3.4 决策表

条件桩	条件项								
C1	Normal	Normal	Normal	Exceed	Exceed	Exceed	SuperExceed	SuperExceed	SuperExceed
C2	Normal	Destory	Super Destory	Normal	Destory	Super Destory	Normal	Destory	Super Destory
动作桩	动作项								
A1				√	√	√	√	√	√
A2	√								
A3		√			√				
A4			√			√	√	√	√

遗漏的需求。

决策表测试法适用于具有以下特征的应用程序：①if-then-else 逻辑突出；②输入变量之间存在逻辑关系；③涉及输入变量子集的计算；④输入与输出之间存在因果关系。

一般情况下，规格说明以决策表形式给出，测试工作可以直接在此基础上进行。另外，要注意，决策表虽然能表达逻辑关系复杂的变量，然而决策表不能体现出变量之间的顺序关系。也就是说，条件(规则)的排列顺序不会也不应影响执行的动作。假如条件项或者动作项存在顺序关系，则并不适合使用决策表这一方式来进行描述。

2. 测试用例的设计

测试用例就是在决策表的基础上完成的，一般情况下，一条规则就是一个测试用例。例如，对于例 3.5，可以设计出 9 个测试用例。设计结果如表 3.5 所示。

表 3.5 测试用例

测试用例	规则号	输入数据	预期输出
Case1	1	VIP1 用户借书 20 天，无磨损	罚款数 0，信誉加 1；如果信誉值为 10，则提高级别为 SVIP
Case2	2	VIP1 用户借书 20 天，磨损	需罚款，信誉减 1；如果信誉值降为 0，则降低级别为 VIP
Case3	3	VIP1 用户借书 20 天，书丢失	需罚款，降级为 VIP
Case4	4	VIP1 用户借书 3 个月，无磨损	需罚款
Case5	5	VIP1 用户借书 3 个月，磨损	需罚款，信誉减 1；如果信誉值降为 0，则降低级别为 VIP
Case6	6	VIP1 用户借书 3 个月，书丢失	需罚款，降级为 VIP
Case7	7	VIP1 用户超期一年，无磨损	需罚款，降级为 VIP
Case8	8	VIP1 用户超期一年，磨损	需罚款，降级为 VIP
Case9	9	VIP1 用户超期一年，书丢失	需罚款，降级为 VIP

利用决策表进行测试用例设计时，应先分析需求，根据需求创建决策表，并完成测试用例设计。当然这样可以做出比较完整的测试用例。但是当需求比较复杂，尤其是条件桩较

多时,测试用例的数量还是相对较大的一个数字,有时是测试人员难以完成的。假如有条件桩数 n,并且每个条件只有两个取值(真、假),那么就会得到 2^n 个规则。当 n 较大时,决策表很烦琐。实际使用决策表时,常常先将它化简。接下来讨论决策表如何进行化简。

决策表的化简是以合并相似规则为目标。即,若表中有两条以上规则具有相同的动作,并且在条件项之间存在极为相似的关系,便可以合并。合并后的条件项用符号"—"表示,说明执行的动作与该条件的取值无关,称为无关条件。

读者可以自行将例3.6的决策表进行化简,看看测试用例数量是否减少了很多。当然化简决策表就会减少测试用例数量,使得测试结果并不如化简前那么完善,但是也不失其代表性。当测试用例数量多时,可以根据测试人员的需求化简决策表。

3. 因果图和决策表

利用3.1.3节介绍的因果图,最终没有得到相应的测试用例。事实上,因果图只是清晰地表达了需求分析的内容。如果要得到测试用例,就必须借助于决策表,也就是需要将因果图转化成决策表。

在因果图中已经分析了"因"和"果","因"和"果"直接可以作为条件桩和动作桩。根据条件桩的取值得到条件项,利用条件项和因果图中原因与结果的关系,可以得到相应的规则,最终生成决策表。

3.2 黑盒测试策略

测试用例的设计方法不是单独存在的,具体到每个测试项目中都会用到多种方法,每种类型的软件各有其特点,因此要针对不同的软件采取不同的黑盒测试方法。在实际测试中,往往是综合使用各种方法才能有效地提高测试效率和测试覆盖率,这就需要认真掌握这些方法的原理,积累更多的测试经验,以有效地提高测试水平。

以下是功能测试部分的各种黑盒测试方法的综合选择策略。

(1) 首先进行等价类划分,包括输入条件和输出条件的等价划分,将无限测试变成有限测试,这是减少工作量和提高测试效率最有效的方法。

(2) 在任何情况下都必须使用边界值分析方法。因为这种方法非常有效,设计出的测试用例发现程序错误的能力最强。

(3) 利用测试人员的经验可以在一些容易出现缺陷的地方追加测试用例,这需要依靠测试工程师的智慧和经验的积累。

(4) 如果程序的功能说明中含有输入条件的组合情况,尤其是各个输入条件之间存在依赖关系,则一开始就可选用因果图法和决策表法。

下面给出一些选取具体测试方法的简单标准:

(1) 如果变量引用的是物理量,可采用边界值分析测试和等价类测试。

(2) 如果变量引用的是逻辑量,可采用等价类测试和决策表测试。

(3) 如果变量是独立的,可采用边界值分析测试和等价类测试。

(4) 如果变量不是独立的,可采用决策表测试。

(5) 如果可保证是单缺陷假设,可采用边界值分析测试和健壮性测试。

(6) 如果可保证是多缺陷假设,可采用边界值分析测试和决策表测试。

(7) 如果程序包含大量例外处理,可采用健壮性测试和决策表测试。

黑盒测试技术是软件测试的主要方法之一。通过黑盒测试可以检查软件的每个功能是否都能正常运行,因此,黑盒测试也是从用户观点和需求的角度进行的测试。3.1 节只介绍了一些常用的黑盒测试方法,当然还有其他的黑盒测试方法,需要读者在以后的学习中逐渐积累。下面对边界值分析、等价类和决策表这 3 种黑盒测试方法进行总结和比较。

边界值分析法是对程序输入或输出的边界值进行测试的一种黑盒测试方法。边界值分析法有其自身的特点。(1)所谓边界值,是基于输入和输出变量的定义域而言的,所以该方法不适合分析数据或逻辑关系。(2)用边界值方法进行测试用例设计,设计方法简单,工作量相对较小,然而生成的测试用例数量比较多,在执行测试时花费的时间会比较长。(3)由于边界值分析方法设计方法简单,所以很容易实现自动化,因此自动化工具对它支持得比较好。

等价类划分法是一种典型的、重要的黑盒测试方法,它将程序所有可能的输入数据划分为若干个等价类,然后从每个等价类中选取具有代表性的数据做为测试用例。等价类划分法的特点是:(1)等价类划分相对于边界值方法,更多地考虑了数据的依赖关系,所以设计方法相对复杂,工作量相对较大。(2)等价类划分所产生的测试用例数量属于中等。由于考虑了数据的关系,所以与边界值方法相比,测试用例数量要少一些,这就会节约测试执行的时间。(3)在标识等价类时需要更多的判断和技巧,等价类标识出以后的处理也是机械的。

决策表是分析和表达多逻辑条件下执行不同操作的情况。决策表最突出的特点是把复杂的问题一一罗列,便于理解,避免遗漏,可以方便地得到测试用例。另外决策表能够考虑到数据的逻辑依赖关系,可以得到完备的测试用例。也正是因为决策表这样的特点,使得这种方法设计工作量较大,不容易利用自动化工具实现。

图 3.8 分别从设计测试用例工作量和得到的测试用例数两个方面对边界值、等价类和决策表这 3 种方法进行对比。由于 3 种方法的复杂程度不同,所以设计测试用例工作量也不相同,其中决策表在测试过程中工作量最大,耗时最长,也相对难以自动化;然而该方法得到的测试用例数却是最少的,也就是执行测试用例时间上是最少的。

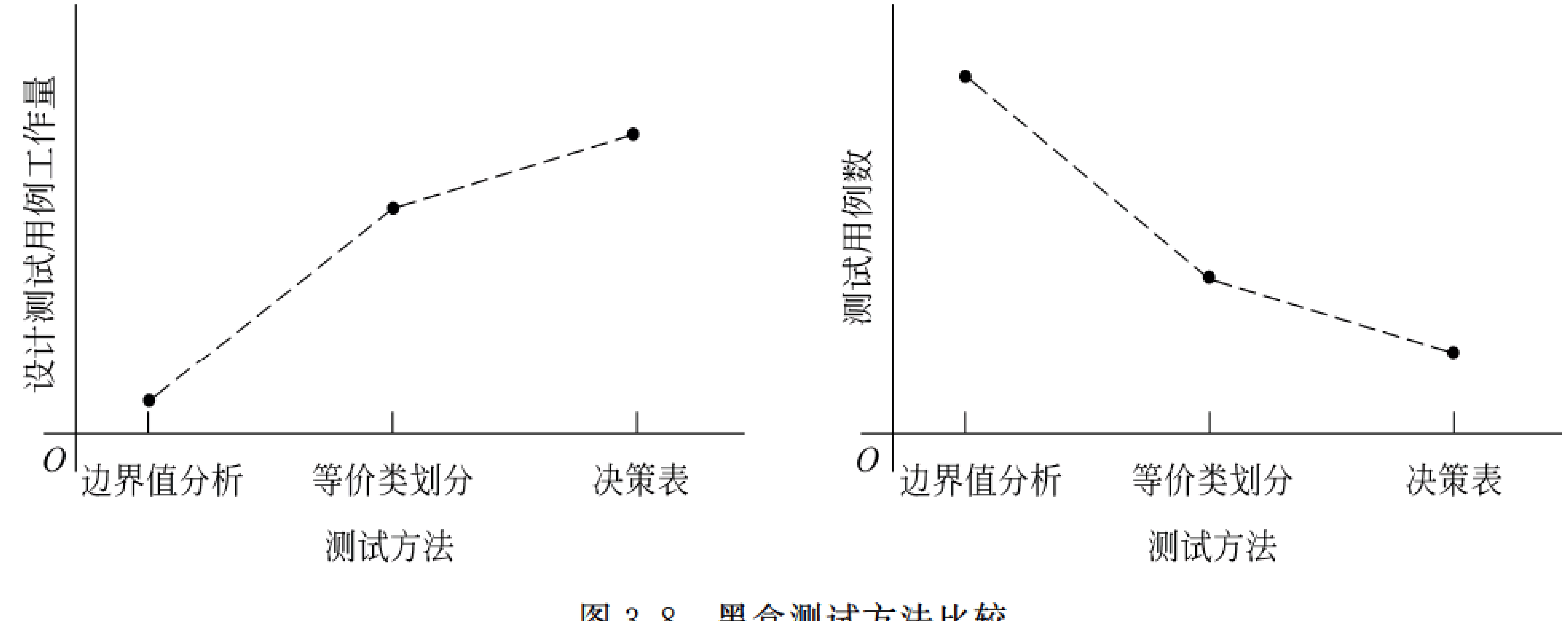


图 3.8 黑盒测试方法比较

在实际应用中,需要结合所有黑盒测试的方法,利用程序结构或者变量的已知属性,选择处理这种属性的方法。在选择使用哪种黑盒测试方法的时候,经常需要考虑以下几个方面:①变量表示物理量还是逻辑量;②在变量之间是否存在依赖关系;③在实际应用中是

否有大量的例外处理，应结合这几种方法的特点，选择合适的黑盒测试方法。

3.3 白盒测试

白盒测试也称结构测试或逻辑驱动测试，它是按照程序内部的结构测试程序，通过测试来检测产品内部动作是否按照设计规格说明书的规定正常进行，检验程序中的每条通路是否都能按预定要求正确工作。这一方法是把测试对象看作一个透明的盒子，测试人员依据程序内部逻辑结构的相关信息，设计或选择测试用例，对程序的所有逻辑路径进行测试，通过在不同点检查程序的状态，确定实际的状态是否与预期的状态一致。

白盒测试的测试方法有代码检查法、静态结构分析法、静态质量度量法、逻辑覆盖法、基本路径测试法、域测试、符号测试、Z路径覆盖、程序变异等测试方法。这些方法又可以划分为两大类：静态测试方法和动态测试方法。其中软件的静态测试不要求在计算机上实际执行被测程序，主要以一些人工的模拟技术对软件进行分析和测试；而软件的动态测试是通过输入一组预先按照一定的测试准则构造的实例数据来动态运行程序，从而达到发现程序错误的目的。在动态分析技术中，最重要的技术是基本路径测试法。下面主要介绍路径测试、数据流测试和逻辑覆盖方法。

3.3.1 路径测试

基本路径测试法是在程序控制流图的基础上，通过分析控制构造的环路（圈）复杂性，导出基本可执行路径集合，从而设计测试用例的方法。并且设计出的测试用例要保证在测试中程序的每个可执行语句至少执行一次。

1. 控制流图

白盒测试是针对软件内部逻辑结构进行的测试，因此，测试人员必须对软件内部结构、各个单元及相互之间的联系和程序的运行过程深入理解。因此，白盒测试是一项庞大、复杂的工作。为了更加突出程序的内部结构，便于测试人员理解源程序，但是又不过多地关注程序中的每一个处理及判断条件，可以对程序流程图进行简化，简化后的程序流程图称为控制流图（control flow graph）。控制流图主要由节点和边构成。一般的控制流图的结构如图3.9所示。控制流图中的每一个带编号的圆被称为一个节点，每个节点可以表示程序中的一条或多条语句，有分支的节点称为判定节点；每一条带箭头的线被称为一条边；由节点和边形成的空间称为区域。图3.9中共有3个区域：$2c3d4b$（区域1），$1a2b4e5f$（区域2）和外围的区域（区域3）。

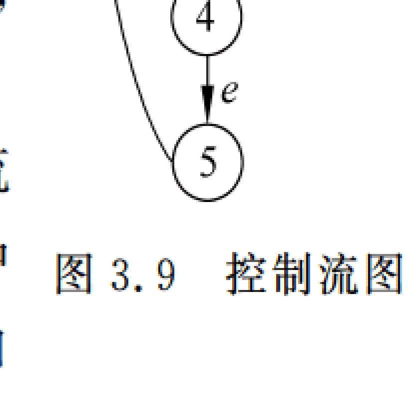


图3.9 控制流图

在路径测试中，先确定程序流程图，再将程序流程图转换为控制流图，转换的原则如下：控制流图中的每一个节点可以表示程序流程图中的以下3种内容：矩形框表示的处理；菱形框表示的两个甚至多个出口判断；多条流线相交的汇合点。程序流程图和转换的控制流图如图3.10所示。

如果判断中的条件表达式是由一个或多个逻辑运算符（OR，AND，NAND，NOR）连接的复合条件表达式，则需要改为一系列只有单条件的嵌套的判断。

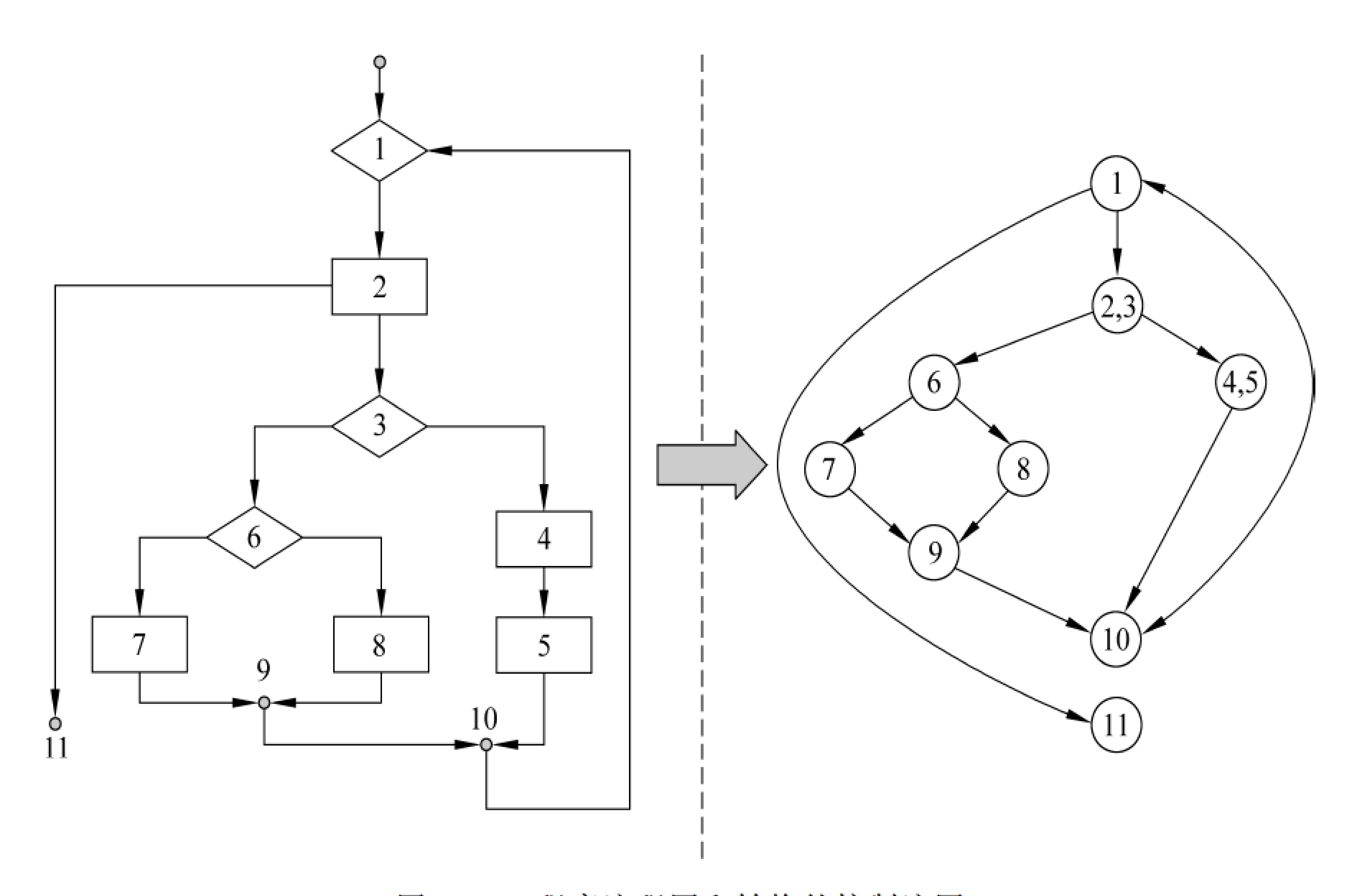


图 3.10　程序流程图和转换的控制流图

例如：

```
1 if a or b
2 x
3 else
4 y
```

对应的控制流图的逻辑结构如图 3.11 所示。

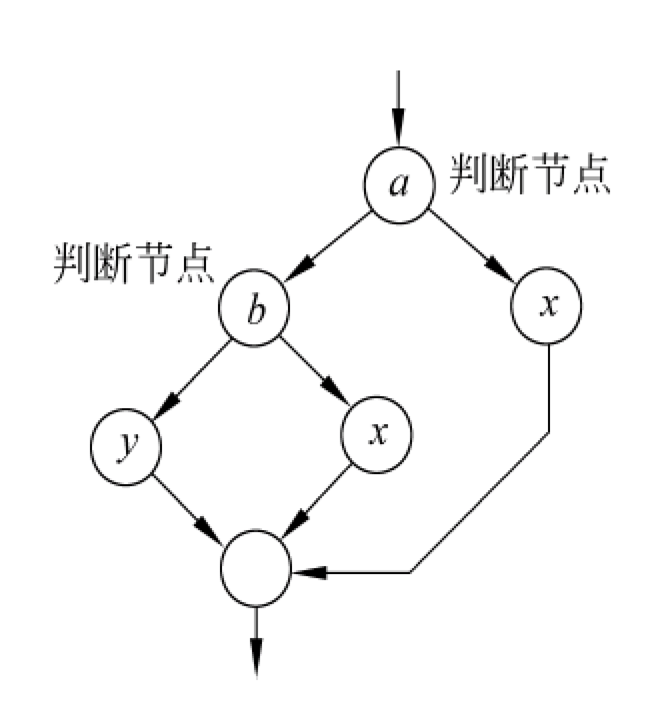


图 3.11　复合条件的逻辑结构

2. 环形(圈)复杂度

环形复杂度是一种为程序逻辑复杂性提供定量测度的软件度量，将该度量用于计算程序的基本的独立路径数目，是确保所有语句至少执行一次所必需的测试用例数量的上界。独立路径必须包含一条在定义之前未曾用到的边。环形复杂度的计算有如下几种方法。

方法一：流图中区域的数量对应于环形的复杂度。

方法二：给定流图 G 的环形复杂度 $V(G)$，定义为 $V(G)=E-N+2$，E 是流图中边的数量，N 是流图中节点的数量。

方法三：给定流图 G 的环形复杂度 $V(G)$，定义为 $V(G)=P+1$，P 是流图 G 中判定节点的数量。

因此，采用以上 3 种方法都可计算出图 3.10 的环形复杂度为 4，图 3.11 的环形复杂度为 3。

3. 路径测试方法应用举例

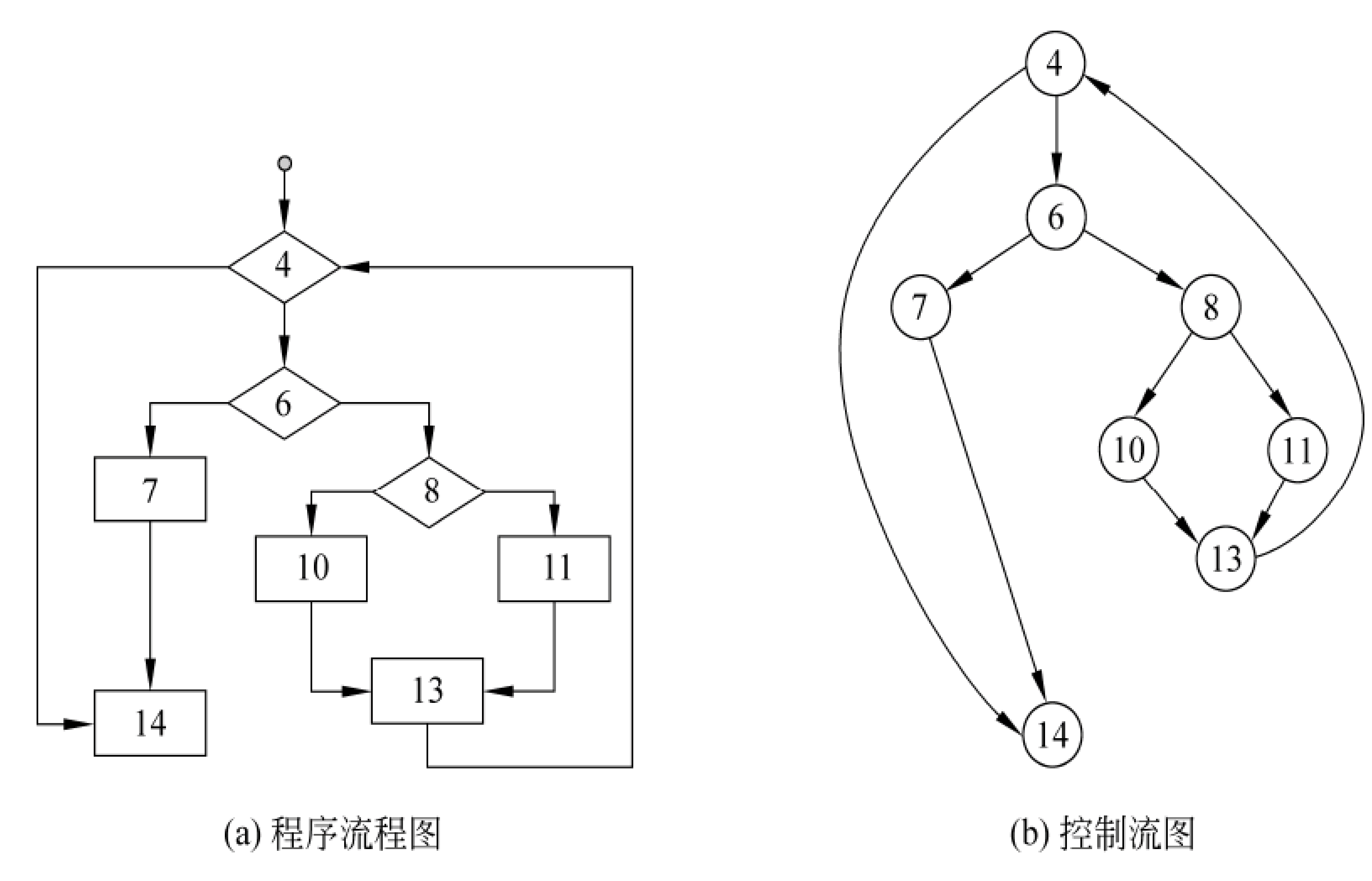
下面以具体的程序为例，介绍路径测试的基本应用。

```
Void Sort(int iRecordNum, int iType)
 1  {
 2      int x=0;
 3      int y=0;
```

```
4       while (iRecordNum-->0)
5       {
6         if(0==iType)
7           x=y+2;
8         else
9           if(1==iType)
10            x=y+10;
11          else
12            x=y+20;
13      }
14  }
```

路径测试方法主要包括 4 个步骤。

第一步,画出程序的控制流图。该程序的控制流图如图 3.12 所示。



(a) 程序流程图　(b) 控制流图

图 3.12　示例程序的程序流程图和控制流图

第二步,计算环形复杂度,并确定独立路径。

根据以上介绍的环形复杂度的计算方法,可计算出 $V(G)=E-N+2=11-9+2=4$。从程序的环形复杂度可导出程序基本路径集合中的独立路径数目,这是确定程序中每个可执行语句至少执行一次所必需的测试用例数目的上界,即独立路径数为 4。因此,该程序的独立路径如下。

路径 1:4-14。

路径 2:4-6-7-14。

路径 3:4-6-8-10-13-4-14。

路径 4:4-6-8-11-13-4-14。

第三步,导出测试用例:根据环形复杂度和程序结构,设计测试用例的数据输入和预期结果,如表 3.6 所示。

第四步,执行测试。

表 3.6 测试用例表

测试用例名称	测试数据	预期结果	测试路径
T1	iRecordNum＝0	x＝0，y＝0	路径 1
T2	iRecordNum＝1 iType＝0	x＝2，y＝0	路径 2
T3	iRecordNum＝1 iType＝1	x＝10，y＝0	路径 3
T4	iRecordNum＝1 iType＝2	x＝20，y＝0	路径 4

3.3.2 数据流测试

路径测试可以测试程序中所有的条件和语句块，但是，这样仍然不能检测出程序中所有的错误。基于数据流的测试主要关注程序中数据的定义和使用，可以作为对基于控制流测试的补充。早期的数据流测试主要关注以下 3 种情况：变量已定义，但从未使用；使用了未定义的变量；变量在使用之前被重复定义。

例如，有如下程序片段：

```
1  int x,y;                    //定义 x,y
2  float z;
3  input(x,y);
4  z=0;
5  if(x!=0)
6     z=z+y;
7  else z=z-y;
8  if(y!=0)
9     z=z/x;
10 else z=z*x;
11 output(z);
```

在程序的第 4 行对变量 z 进行初始化，在后面的语句中根据变量 x 和 y 的情况修改 z 的值。这里可以采用两个测试用例，x、y 和 z 变量的取值分别是：T1(0,0,0.0)，T2(1,1,0.0)。执行 T1 的路径为先执行第 7 行修改 z 的值，再执行第 10 行；执行 T2 的路径为先执行第 6 行修改 z 的值，再执行第 9 行。同样，z 在第 6、7、9、10 行也被定义了，但是，程序的执行可能是：在第 7 行被定义后，在第 9 行被使用了，那么，这就可能出现“除零”的情况（如 x＝0，y＝1）。如果采用数据流测试就可以发现程序中类似的错误。

1. 定义/使用测试

首先要明确一个假设，数据流的假设还是和路径的假设一致：程序 P 的程序图（有向图）为单入口、单出口，并且不允许有从某个节点到其自身的边。

DEF(v,n)：定义节点，变量 v 在节点 n（语句片段）处定义，包括输入语句、赋值语句（等号左侧）、循环语句及过程调用都是定义节点的例子，如果执行这些语句，变量的值往往会发生变化。如：int x；x＝y＋z，这两条语句都是对 x 的定义。

USE(v,n)：使用节点，变量 v 在节点 n 处被使用，包括输出语句、赋值语句(等号右侧)、条件语句、循环控制语句和过程调用语句都是节点的使用语句，如果执行这类语句，值不会被改变。如：System.out.println(x)，该语句中使用了变量 x。

P-use：当一个变量被用在分支语句的条件表达式中(如 if 和 while 语句)，则称为变量的 P-use。其中的 P 表示“谓词”。如：if(z＜0){…}。其中的条件表达式是变量 z 的 P-use。

C-use：如果一个变量被用在赋值语句的表达式和输出语句中，被当作参数传递给调用函数，或被用在下标表达式中，则称为变量的 C-use。其中，C 表示“计算”。如：y＝x＋1；function(x)，这两条语句是变量 x 的 C-use。

DU-path：定义使用路径，开始节点是 DEF(v,n)，结束节点是 USE(v,n)的路径。

DC-path：定义清除路径，当开始节点和结束节点中间没有其他的定义节点时为清除路径。

对于上面的程序，给出其定义节点和使用节点，如表 3.7 所示。

表 3.7　示例程序中变量的定义节点和使用节点

变量	定义节点	使用节点
x	3	5,9,10
y	3	6,7,8
z	4,6,7,9,10	6,7,9,10,11

表 3.8、表 3.9 和表 3.10 分别列出了变量 x、y 和 z 的定义使用路径(DU-path)，并标出对应的路径是否为定义清除路径(DC-path)。

表 3.8　变量 x 的定义使用路径

DU-path(开始节点，结束节点)	是否为 DC-path
3,5	Y
3,9	Y
3,10	Y

表 3.9　变量 y 的定义使用路径

DU-path(开始节点，结束节点)	是否为 DC-path
3,6	Y
3,7	Y
3,8	Y

在对变量 x、y 和 z 的分析中发现，变量 x 和 y 的定义使用路径比较简单，而且其定义使用路径也都是定义清除路径；而变量 z 则比较复杂，定义使用路径中出现了部分非定义清除路径，如开始节点和结束节点分别为 4 和 9 的路径为 p1＝＜4,5,6,8,9＞和 p2＝＜4,5,7,8,9＞，变量 z 在节点 4 被定义之后，有可能在节点 6 或 7 被重新定义之后，在节点 9 再使用该变量，因此以节点 4 开始和以节点 9 结束的这两条路径 p1 和 p2 都是非定义清除路径。在定义使用测试中，应该重点关注非定义清除路径，这样就可以发现类似“除零”的错误。

表 3.10　变量 z 的定义使用路径

DU-path(开始节点,结束节点)	是否为 DC-path	DU-path(开始节点,结束节点)	是否为 DC-path
4,6	Y	7,10	Y
4,7	Y	7,11	N
4,9	N	9,6	不可行
4,10	N	9,7	不可行
4,11	N	9,9	Y
6,6	Y	9,10	不可行
6,7	不可行	9,11	Y
6,9	Y	10,6	不可行
6,10	Y	10,7	不可行
6,11	N	10,9	不可行
7,6	不可行	10,10	Y
7,7	Y	10,11	Y
7,9	Y		

2. 程序片测试

程序片也叫程序切片，是一种程序分析和理解技术。它通过把程序减少到只包含与某个特定计算相关的那些语句来分析程序。其概念最早是1979年由Mark Weiser提出来的。他观察到，程序员在调试过程中脑海中就有关于程序的某种抽象，人们在调试一个程序时总是从错误语句开始，并沿着依赖关系跟踪到它影响的程序部分。程序片的发展基本成熟，在理论和应用方面的研究均取得了可喜的进展，特别是在程序的调试、测试、分解和集成、软件维护、代码理解以及逆向工程等领域具有广泛的应用。

程序片是确定或影响某个变量在程序某个点上的取值的一组程序语句。典型的程序分片算法有Weiser的基于数据流方程的算法、无定型分片算法、Bergeretti的基于信息流关系的算法、基于程序依赖图的图形可达性算法、基于波动图的算法、参数化程序分片算法、并行分片算法和面向对象的分层分片算法等。在众多程序分片算法中，Weiser的基于数据流方程的算法是程序分片的基础，下面给出片的定义。

定义1：给定一个程序 P 和 P 中的一个变量集合 V，变量集合 V 在语句 n 上的一个片记做 $S(V,n)$，是 P 中对 V 中的变量值做出贡献的所有语句集合。

定义2：给定一个程序 P 和一个给出语句及语句片段编号的程序图 $G(P)$，以及 P 中的一个变量集合 V，变量集合 V 在语句片段 n 上的一个片记做 $S(V,n)$，是 P 中在 n 以前对 V 中的变量值作出贡献的所有语句片段编号的集合。

假设片 $S(V,n)$ 是一个变量的片，即 V 只有一个元素 v。如果语句片段 n 是 v 的一个定义节点，则 n 包含在该片中；如果语句片段 n 是 v 的一个使用节点，则 n 不包含在该片中。

其他变量的谓词使用和计算使用，要包含其执行会影响变量 v 取值的节点。

切片算法的基本过程为：首先寻找语句 n 的变量 v 所直接数据依赖或控制依赖的节点；然后寻找这些新节点所直接数据依赖或控制依赖的节点；一直重复下去，直到没有新节点加进来为止；最后将这些节点按源程序的语句顺序排列，即为程序 P 的关于语句 n 的片 S。

下面就本节开头的程序片段为例来分析程序片的划分。

变量 x 上的片如下：

S1：$S(x,3)=\{3\}$

S2：$S(x,5)=\{3\}$

变量 y 上的片如下：

S3：$S(y,3)=\{3\}$

S4：$S(y,8)=\{3\}$

变量 z 上的片如下：

S5：$S(z,4)=\{4\}$

S6：$S(z,6)=\{3,4,5,6\}$

S7：$S(z,7)=\{3,4,5,7\}$

S8：$S(z,9)=\{3,4,5,6,7,8,9\}=\{S(z,6),7,8,9\}$

S9：$S(z,10)=\{3,4,5,6,7,8,10\}=\{S(z,6),7,8,10\}$

S10：$S(z,11)=\{3,4,5,6,7,8,9,10\}=\{S(z,9),10\}$

将程序进行分片之后，可以集中注意力于感兴趣的程序部分，排除不相关的部分。程序分片的基本原则为：对所有的赋值定义节点建立片；对谓词使用节点建立片；对每一个变量建立片；不要在不出现变量的语句节点上建立片；使片具有可编译性。

程序片测试也是用于排除程序中的错误。例如，有关于变量 v 的两个程序片 $S(v,5)$ 和 $S(v,8)$，假设第 5 行和第 8 行之间没有任何常量对 v 进行赋值，那么 $S(v,8)=(S(v,5),6,7,8)$，这里假设第 6、7、8 行影响了 v 的值，那么很自然地会想到，如果第 5 行之前的程序片 $S(v,5)$ 中的变量 v 没有任何问题，而第 8 行的程序片出现了问题，那么变量 v 的异常必然在 $S(v,8)$ 至 $S(v,5)$ 这段程序片上。因此程序片能够很快定位出异常。

3.4 逻辑覆盖

逻辑覆盖是通过对程序逻辑结构的遍历实现对程序的覆盖，它是一系列测试过程的总称，这组测试过程逐渐进行越来越完整的通路测试。根据覆盖目标的不同和覆盖源程序语句的详尽程度，逻辑覆盖可分为语句覆盖、判定(分支)覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖和路径覆盖。

3.4.1 语句覆盖

语句覆盖是选择足够多的测试数据，使得程序中的每个可执行语句至少执行一次。

示例程序片段如下：

```
If(A>1&&B=0)then
    x=x/A;
If(A=2||x>1)then
    x=x++;
```

程序流程图如图 3.13 所示。

要做到语句覆盖，可以选择测试用例 A=2，B=0，x=4，则程序按照路径 ade 执行，即可做到语句覆盖。如果选择测试用例 A=2，B=1，x=4，则程序按照路径 abe 执行，而语句 x=x/A 没有被执行，因此没有做到语句覆盖。

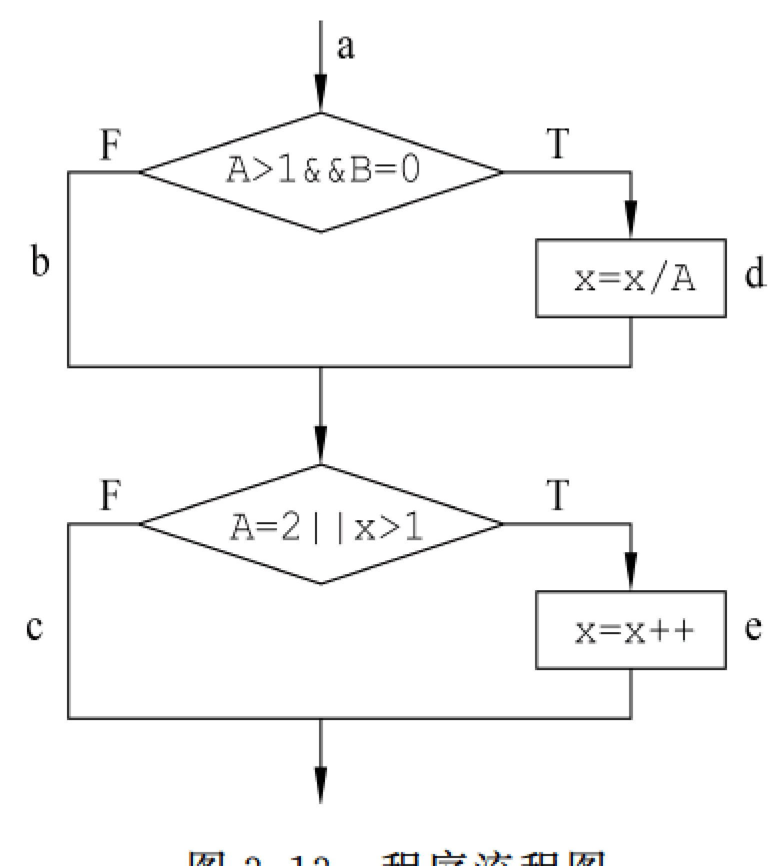


图 3.13 程序流程图

语句覆盖的优点是可以很直观地从源代码得到测试用例，无须细分每条判定表达式。这种测试方法的不足是仅仅针对程序逻辑中显式存在的语句，但对于隐藏的条件是无法测试的。如在多分支的逻辑运算中无法全面地考虑，语句覆盖是最弱的逻辑覆盖。

3.4.2 判定覆盖

判定覆盖是设计足够多的测试用例，使得程序中的每一个判断至少获得一次“真”和一次“假”，即使得程序流程图中的每一个真、假分支至少被执行一次。以上程序示例中，要做到判定覆盖，可选择的测试用例如表 3.11 所示。

表 3.11 判定覆盖的测试用例分析

测试用例	变量取值(A,B,x)	执行路径
t1	2,0,4	ade
t2	1,0,1	abc

由判定覆盖的定义及以上测试用例可以看到，判定覆盖同时还做到了语句覆盖。因此，判定覆盖具有比语句覆盖更强的测试能力。同样也具有和语句覆盖一样的简单性，无须细分每个判定就可以得到测试用例。但是，往往大部分的判定语句由多个逻辑条件组合而成，若仅仅判断其整个最终结果，而忽略每个条件的取值情况，必然会遗漏部分测试路径。因此，判定覆盖仍是弱的逻辑覆盖。

3.4.3 条件覆盖

条件覆盖要求设计足够多的测试用例，使得程序中每个判定表达式中的每个条件至少获得一次“真”和一次“假”。对于以上的程序示例分析如下。

第一个判定表达式：

$A>1$ 取真值，即 $A>1$，记为 $T1$；

$A>1$ 取假值，即 $A\leqslant 1$，记为 $\sim T1$；

$B=0$ 取真值，即 $B=0$，记为 $T2$；

$B=0$ 取假值,即 $B\neq0$,记为～$T2$。

第二个判定表达式:

$A=2$ 取真值,即 $A=2$,记为 $T3$;

$A=2$ 取假值,即 $A\neq2$,记为～$T3$;

$x>1$ 取真值,即 $x>1$,记为 $T4$;

$x>1$ 取假值,即 $x\leqslant1$,记为～$T4$。

通过以上分析,可设计满足条件覆盖的测试用例如表 3.12 和表 3.13 所示。

表 3.12 条件覆盖的测试用例分析(1)

测试用例	变量取值(A,B,x)	执行路径	覆盖条件
t1	2,0,4	ade	$T1,T2,T3,T4$
t2	1,0,1	abc	～$T1,T2,$～$T3,$～$T4$
t3	2,1,2	abe	$T1,$～$T2,T3,T4$

表 3.13 条件覆盖的测试用例分析(2)

测试用例	变量取值(A,B,x)	执行路径	覆盖条件
t1	1,0,1	abc	～$T1,T2,$～$T3,$～$T4$
t2	2,1,2	abe	$T1,$～$T2,T3,T4$

由以上分析可以看到,表 3.12 中的测试用例在做到条件覆盖的同时,也做到了判定覆盖;而表 3.13 中的测试用例虽然做到了条件覆盖,但是没有做到判定覆盖,第一个判定的真分支没有被覆盖到。因此,可以看到覆盖了条件的测试用例不一定会覆盖判定。条件覆盖增加了对条件判定情况的测试,增加了测试路径,但是不一定包含判定覆盖。因此,条件覆盖只能保证每个条件至少有一次为真,而不考虑所有的判定结果。

3.4.4 判定/条件覆盖

判定/条件覆盖要求使得判断中每个条件的所有可能至少出现一次,并且每个判断本身的判定结果也至少出现一次。

以上程序示例中包含两个判断,每个判断又由两个条件组成,因此,对于程序的分析如下:

$A>1,B=0$,记为 $T1,T2$;

$A>1,B\neq0$,记为 $T1,$～$T2$;

$A\leqslant1,B=0$,记为～$T1,T2$;

$A\leqslant1,B\neq0$,记为～$T1,$～$T2$;

$A=2,x>1$,记为 $T3,T4$;

$A=2,x\leqslant1$,记为 $T3,$～$T4$;

$A\neq2,x>1$,记为～$T3,T4$;

$A\neq2,x\leqslant1$,记为～$T3,$～$T4$。

判定/条件覆盖的测试用例分析如表 3.14 所示。

表 3.14 判定/条件覆盖测试用例分析

测试用例	变量取值(A,B,x)	执行路径	覆盖条件
t1	2,0,4	ade	$T1,T2,T3,T4$
t2	1,1,1	abc	$\sim T1,\sim T2,\sim T3,\sim T4$

由以上测试用例分析可以看到,该组测试用例同时满足判定覆盖和条件覆盖。但是该覆盖准则的缺点是未考虑条件的组合情况。从表面的分析来看,它测试了所有条件的取值,但实际往往因为一些条件而掩盖了另一些条件。对于条件表达式 $A>1\&\&B=0$ 来说,只要 $A>1$ 的测试为真,才需测试 $B=0$ 的值来确定此表达式的值;但是若 $A>1$ 的测试值为假时,不需再测 $B=0$ 的值就可确定此表达式的值为假,因而 $B=0$ 没有被检查。同理,对于 $A=2\parallel x>1$ 这个表达式来说,只要 $A=2$ 测试结果为真,不必测试 $x>1$ 的结果就可确定表达式的值为真。所以对于判定/条件覆盖来说,逻辑表达式中的错误不一定能够检查出来。

3.4.5 条件组合覆盖

条件组合覆盖要求设计足够多的测试用例,使得每个判定中条件的各种可能组合都至少出现一次。

以上程序示例中包含两个判断,两个判断中共有 4 个条件,条件的各种可能组合情况有 8 种:

(1) $A>1,B=0$ 属第一个判断的取真分支;
(2) $A>1,B\neq0$ 属第一个判断的取假分支;
(3) $A\leqslant1,B=0$ 属第一个判断的取假分支;
(4) $A\leqslant1,B\neq0$ 属第一个判断的取假分支;
(5) $A=2,x>1$ 属第二个判断的取真分支;
(6) $A=2,x\leqslant1$ 属第二个判断的取真分支;
(7) $A\neq2,x>1$ 属第二个判断的取真分支;
(8) $A\neq2,x\leqslant1$ 属第二个判断的取假分支。

条件组合覆盖的测试用例分析如表 3.15 所示。

表 3.15 条件组合覆盖测试用例分析

测试用例	变量取值(A,B,x)	执行路径	覆盖组合情况
t1	2,0,4	ade	(1),(5)
t2	2,1,1	abe	(2),(6)
t3	1,0,2	abe	(3),(7)
t4	1,1,1	abc	(4),(8)

从以上的分析来看,条件组合覆盖满足判定覆盖、条件覆盖和判定/条件覆盖准则,是前述几种覆盖标准中发现错误能力最强的。但是,从测试用例数量来看,这种覆盖准则线性增加了测试用例的数量。

3.4.6 几种覆盖准则之间的区别及关系

表 3.16 列出了几种覆盖准则在发现错误的能力及测试用例应满足的条件上的区别。

表 3.16 几种覆盖准则比较

覆盖标准	发现错误的能力	程序结构举例	测试用例应满足条件
语句覆盖	1(最弱)	T A^B F	A^B=T
判定覆盖	2	T A^B F	A^B=T A^B=F
条件覆盖	3	T A^B F	A=T A=F B=T B=F
判定/条件覆盖	4	T A^B F	A^B=T A^B=F A=T A=F B=T B=F
条件组合覆盖	5(最强)	T A^B F	A=T^B=T A=T^B=F A=F^B=T A=F^B=F

几种覆盖准则之间的关系如图 3.14 所示。

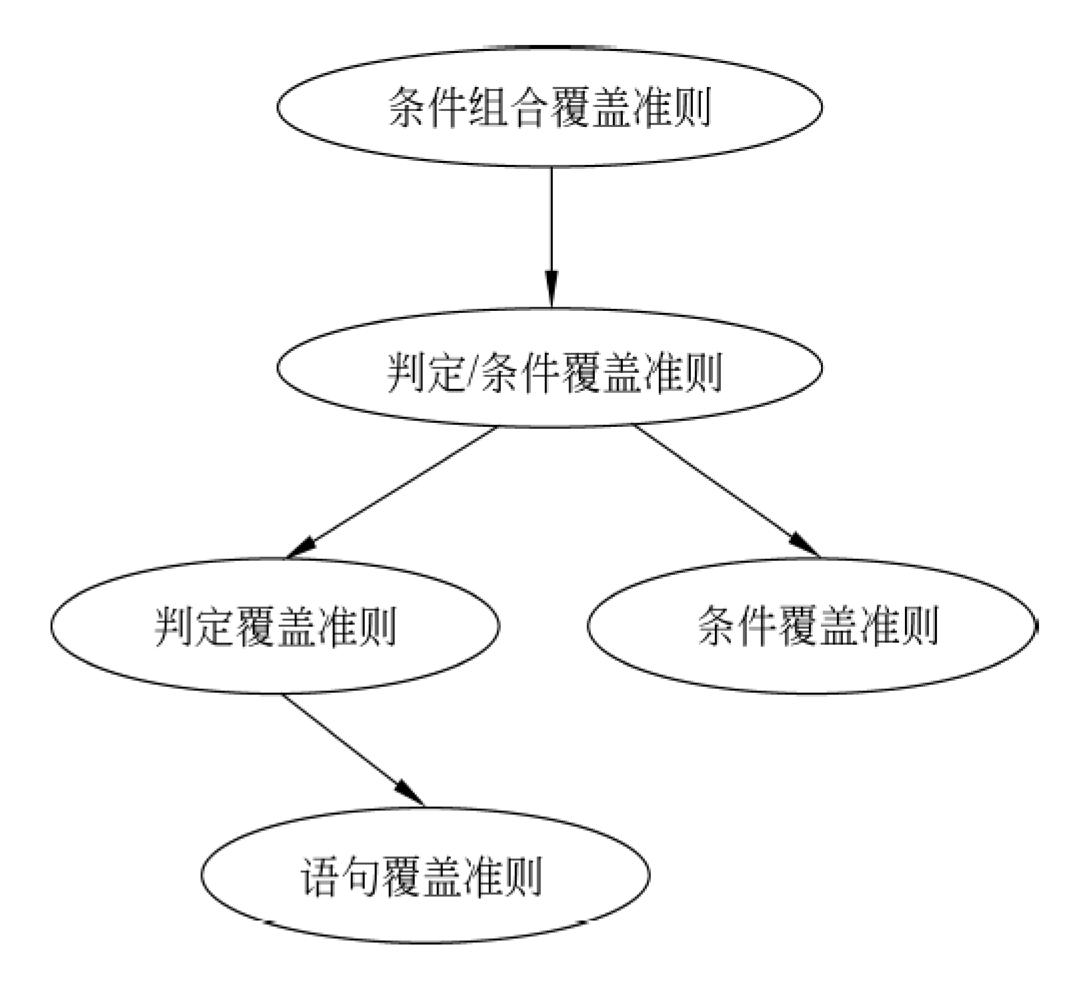


图 3.14 几种覆盖准则之间的关系

可以看到，几种覆盖准则中，语句覆盖准则是最弱的一种，条件组合覆盖准则是最强的。并且可以看到，满足判定覆盖准则的测试用例一定包含了语句覆盖准则，判定覆盖准则和条件覆盖准则之间没有必然的包含关系，而满足判定/条件覆盖准则的测试用例同时满足判定覆盖准则和条件覆盖准则，条件组合覆盖准则包含了判定/条件覆盖准则。

3.5 白盒测试策略

在软件开发过程的不同阶段都可能需要进行白盒测试，根据实际的开发情况，可有选择地使用下面的白盒测试策略。

3.5.1 桌前检查

桌前检查是在程序员实现特定功能之后，进行单元测试之前，对源代码进行的初步检查。该项工作的参与人员为开发人员，重点检查编码、语句的使用等是否符合编码规范，并根据编码规范调整自己的代码以使之符合要求。

3.5.2 单元测试

单元测试也称为模块测试，在传统的结构化程序中，以一个函数、过程为一个单元；在面向对象编程过程中，一般将类作为单元进行测试。该项工作的参与人员为专门的白盒测试人员。可采用白盒测试和黑盒测试相结合的方法。

3.5.3 代码评审

代码评审是在编码初期或编写过程中采用的一种有同行参与的评审活动。该项工作需要所有开发小组共同参与，通过大家共同阅读代码，或者由程序编写者讲解代码，其他同行边听边分析问题的方法，共同查看程序，可以找出问题，使大家的代码风格一致或遵守编码规范。

3.5.4 同行评审

在同行评审中，由软件产品创建者的同行检查该工作产品，识别产品的缺陷，改进产品的不足。同行评审主要用于检验工作产品是否正确地满足了以往的工作产品中建立的规范，如需求或设计文档；识别工作产品相对于标准的偏差，包括可能影响软件可维护性的问题；向创建者提出改进建议；促进参与者之间的技术交流和学习等。根据 CMM 标准，该项工作的参与人员为程序员、设计师、单元测试工程师、维护者、需求分析师和编码标准专家，至少需要开发人员、测试人员和设计师。

3.5.5 代码走查

代码走查由测试小组组织，或者由专门的代码走查小组进行代码走查，这时需要开发人员提交有关的资料文档和源代码给走查人员，并进行必要的讲解。代码走查往往根据代码检查单来进行，代码检查单常常是根据编码规范总结出来的一些条目，目的是检查代码是否

按照编码规范来编写的。当然,代码走查的最终目的还是为了发现代码中潜在的错误和缺陷。该项工作的参与者为测试人员。代码走查速度一般建议为:汇编代码与C语言代码150行/小时,C++/Java代码200~300行/小时。

3.5.6 静态分析

静态分析通常需要辅助工具支持,通过提取代码信息,进行统计,根据统计结果对源代码进行质量评估。代码规则检查也是静态分析的一个方面。该项工作的参与人员为测试小组。

桌面检查、代码走查和代码审查同时属于代码检查的方式。代码检查是发现错误和缺陷最有效的手段之一,通常能发现30%~70%的逻辑设计和编码缺陷。可以发现的问题包括声明或引用错误、函数/方法参数错误、语句不可达错误、数组越界错误、控制流错误、界面错误和输入输出流错误等。

实际测试过程中,可能会因为项目不同而采取不同的测试策略,组织形式和参加人员也可能不同,但白盒测试不会超出这些范畴。

3.6 案例分析——佣金问题的数据流测试分析

3.6.1 问题描述及分析

佣金问题是数据流测试中广泛使用的一个经典案例,问题描述如下:美国亚利桑那州境内的步枪销售商销售密苏里州军械厂制造的步枪枪机(lock)、枪托(stock)和枪管(barrel),机枪卖45美元,枪托卖30美元,枪管卖25美元。销售商每月至少要售出一支完整的步枪,而生产商的生产能力限制销售商在一个月内最多可销售70个枪机、80个枪托和90个枪管。每走访过一个城镇之后,销售商都要给密苏里军械厂发一封电报,汇报在这一城镇中销售枪机、枪托和枪管的数量。销售商月末会再发一封很短的电报,通知“**个枪机售出”。这样军械厂就知道当月的销售活动已经结束,可计算销售商应得的佣金了。佣金计算方法如下:销售总额1000美元以下(含1000美元)部分的佣金为10%,1000~1800美元之间部分的佣金为15%,超过1800美元的部分的佣金为20%。佣金程序应该生成月份销售报表,汇总出该销售商当月枪机、枪托和枪管的销售总量、总销售额以及应得佣金。

这个佣金程序可分为3个部分:输入数据处理部分,验证输入数据的有效性(同三角形问题和NextDate函数一样);销售额统计计算部分;佣金计算部分。此处省略了对输入数据有效性的验证,采用了常用的条件循环语句While来模拟对电报的处理。该问题的伪码描述如下:

```
Program Commission (INPUT, OUTPUT)
Dim locks, stocks, barrels As Integer
Dim lockPrice, stockPrice, barrelPrice As Real
Dim totalLocks, totalStocks, totalBarrels As Integer
Dim lockSales, stockSales, barrelSales As Real
Dim sales, commission As Real
lockPrice = 45.0
```

```
stockPrice = 30.0
barrelPrice = 25.0
totalLocks = 0
totalStocks = 0
totalBarrels = 0
Input (locks)
While NOT(locks = -1)  'Input device uses -1 to indicate end of data
  Input (stocks, barrels)
  totalLocks = totalLocks + locks
  totalStocks = totalStocks + stocks
  totalBarrels = totalBarrels + barrels
  Input (locks)
EndWhile
Output ("Locks sold: ", totalLocks)
Output ("Stocks sold: ", totalStocks)
Output ("Barrels sold: ", totalBarrels)
lockSales = lockPrice * totalLocks
stockSales = stockPrice * totalStocks
barrelSales = barrelPrice * totalBarrels
sales = lockSales + stockSales + barrelSales
Output ("Total sales: ", sales)
If (sales > 1800.0)
  Then
    commission = 0.10 * 1000.0
    commission= commission + 0.15 * 800.0
    commission= commission + 0.20 * (sales- 1800.0)
  Else If (sales > 1000.0)
    Then
      commission = 0.10 * 1000.0
      commission= commission + 0.15 * (sales- 1000.0)
    Else
      commission = 0.10 * sales
    EndIf
EndIf
Output ("Commission is $", commission)
End Commission
```

3.6.2 佣金问题的定义/使用测试

佣金问题的程序图如图 3.15 所示。

佣金问题中,变量的定义使用节点如表 3.17 所示。对应变量的定义使用路径如表 3.18 至表 3.22 所示。

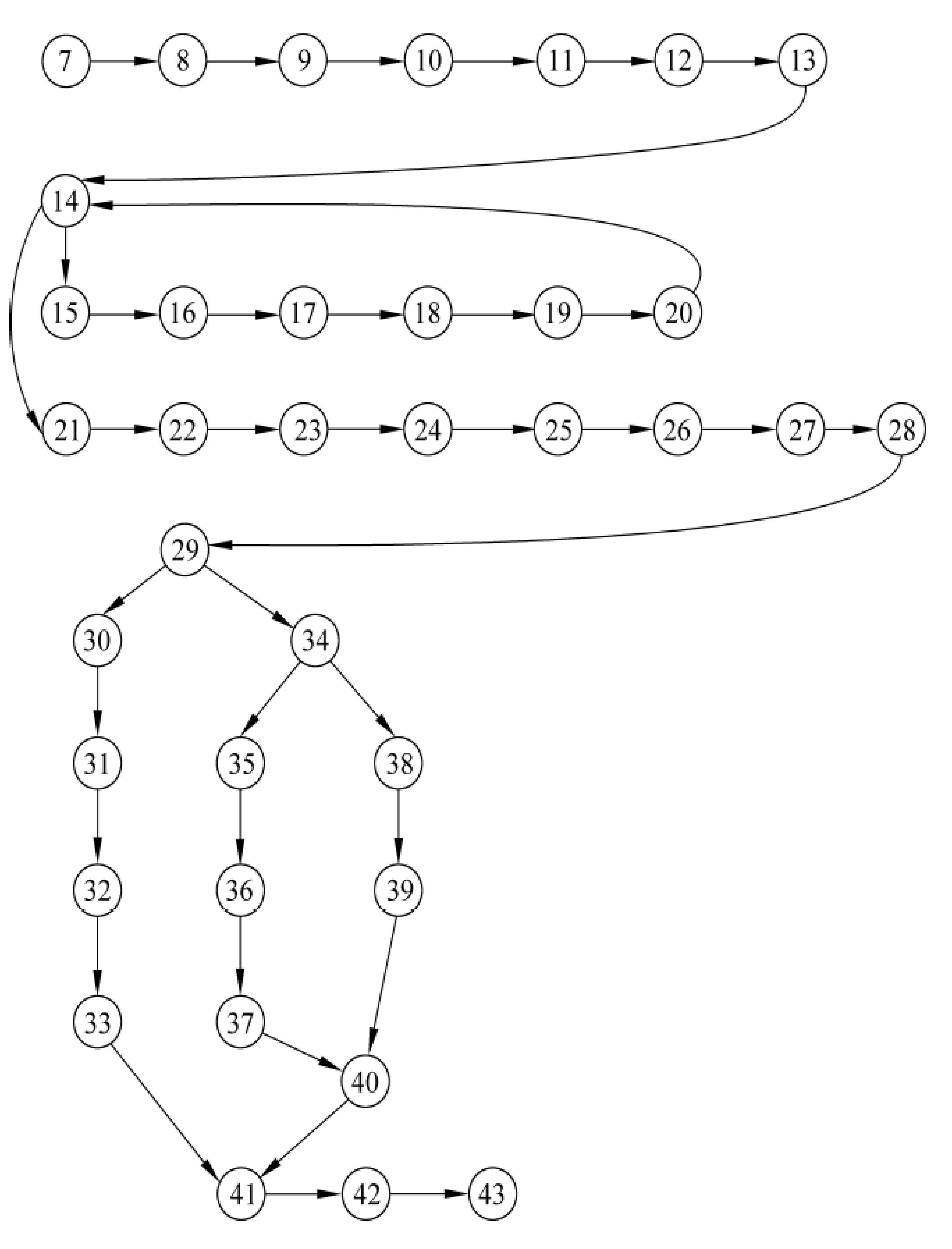


图 3.15 佣金问题的程序图

表 3.17 变量的定义使用节点

变量	定义节点	使用节点
lockPrice	7	24
stockPrice	8	25
barrelPrice	9	26
totalLocks	10,16	16,21,24
totalStocks	11,17	17,22,25
totalBarrels	12,18	18,23,26
locks	13,19	14,16
stocks	15	17
barrels	15	18
lockSales	24	27
stockSales	25	27
barrelSales	26	27
sales	27	28,29,33,34,37,38
commission	31,32,33,36,37,38	32,33,37,41

表 3.18　lockPrice、stockPrice 和 barrelPrice 的定义使用路径

变量	路径节点	是否定义清除路径
lockPrice	7,…,24	是
stockPrice	8,…,25	是
barrelPrice	9,…,26	是

表 3.19　locks、stocks 和 barrels 的定义使用路径

变量	路径节点	是否定义清除路径
locks	13,14	是
locks	13,14,15,16	是
locks	19,20,14	是
locks	19,20,14,15,16	是
stocks	15,…,17	是
barrels	15,…,18	是

表 3.20　totalLocks 的定义使用路径

路径节点	是否定义清除路径	路径节点	是否定义清除路径
10,16	是	16,16	是
10,21	否	16,21	是
10,24	否	16,24	是

表 3.21　sales 的定义使用路径

路径节点	是否定义清除路径	路径节点	是否定义清除路径
27,28	是	27,28,29,34	是
27,29	是	27,28,29,34,35,36,37	是
27,33	是	27,28,29,38	是

表 3.22　commission 的定义使用路径

路径节点	是否定义清除路径	路径节点	是否定义清除路径
31,32	是	32,37	—
31,33	否	32,41	否
31,37	—	33,32	—
31,41	否	33,33	是
32,32	是	33,37	—
32,33	是	33,41	是

续表

路径节点	是否定义清除路径	路径节点	是否定义清除路径
36,32	—	37,37	是
36,33	—	37,41	是
36,37	是	38,32	—
36,41	否	38,33	—
37,32	—	38,37	—
37,33	—	38,41	是

3.6.3 佣金问题的程序片测试

根据对佣金问题的详细分析,可得到程序中对应的变量在节点中的程序片的划分。其中,变量 locks 上的片划分为:

S1:S(locks,13)={13};

S2:S(locks,14)={13,14,19,20};

S3:S(locks,16)={13,14,16,19,20};

S4:S(locks,19)={19}。

变量 stocks 和 barrels 上的片划分为:

S5:S(stocks,15)={13,14,15,19,20};

S6:S(stocks,17)={13,14,15,17,19,20};

S7:S(barrels,15)={13,14,15,19,20};

S8:S(barrels,18)={13,14,15,18,19,20}。

变量 totalLocks、totalStocks 和 totalBarrels 上的片划分为:

S9:S(totalLocks,10)={10};

S10:S(totalLocks,16)={10,13,14,16,19,20};

S11:S(totalLocks,21)={10,13,14,16,19,20};

S12:S(totalStocks,11)={11};

S13:S(totalStocks,17)={11,13,14,15,17,19,20};

S14:S(totalStocks,22)={11,13,14,15,17,19,20};

S15:S(totalBarrels,12)={12};

S16:S(totalBarrels,18)={12,13,14,15,18,19,20};

S17:S(totalBarrels,23)={12,13,14,15,18,19,20}。

变量 lockPrice、stockPrice、barrelPrice、lockSales、stockSales 和 barrelSales 上的片的划分为:

S18:S(lockPrice,24)={7};

S19:S(stockPrice,25)={8};

S20:S(barrelPrice,26)={9};

S21:S(lockSales,24)={7,10,13,14,16,19,20,24};

S22：S(stockSales,25)={8,11,13,14,15,17,19,20,25}；

S23：S(barrelSales,26)={9,12,13,14,15,18,19,20,26}。

变量 sales 上的片的划分为：

S24：S(sales,27)={7,8,9,10,11,12,13,14,15,16,17,18,19,20,24,25,26,27}；

S25：S(sales,28)={7,8,9,10,11,12,13,14,15,16,17,18,19,20,24,25,26,27}；

S26：S(sales,29)={7,8,9,10,11,12,13,14,15,16,17,18,19,20,24,25,26,27}；

S27：S(sales,33)={7,8,9,10,11,12,13,14,15,16,17,18,19,20,24,25,26,27}；

S28：S(sales,34)={7,8,9,10,11,12,13,14,15,16,17,18,19,20,24,25,26,27}；

S29：S(sales,37)={7,8,9,10,11,12,13,14,15,16,17,18,19,20,24,25,26,27}；

S30：S(sales,38)={7,8,9,10,11,12,13,14,15,16,17,18,19,20,24,25,26,27}。

变量 commission 上的片的划分为：

S31：S(commission,31)={31}；

S32：S(commission,32)={31,32}；

S33：S(commission,33)={7,8,9,10,11,12,13,14,15,16,17,18,19,20,24,25,26,27,29,30,31,32,33}；

S34：S(commission,36)={36}；

S35：S(commission,37)={7,8,9,10,11,12,13,14,15,16,17,18,19,20,24,25,26,27,36,37}；

S36：S(commission,38)={7,8,9,10,11,12,13,14,15,16,17,18,19,20,24,25,26,27,29,34,38}；

S37：S(commission,41)={7,8,9,10,11,12,13,14,15,16,17,18,19,20,24,25,26,27,29,30,31,32,33,34,35,36,37,38}。

3.7 面向对象的测试用例设计

传统的软件测试是一个自底向上的过程，首先从单元测试开始，集中测试程序最小的可编译单位（函数、模块或进程），单元测试确定完成后，再进行集成测试，之后逐步进入系统测试，直到进行验收测试。不管采用什么样的测试方式，基本还是遵从这样的一个模式。显然，对于测试面向对象的程序，这样的模式不再适用。

“面向对象”这一概念就是相对于“面向过程”而提出来的，是传统结构化的一种本质上的创新，体现了“对象”这个整体的概念，已经不可能用功能细化的观点来分析程序。因此，传统的测试方法也不完全适用于这种编程模式。面向对象技术是一种全新的软件开发技术，正逐渐代替被广泛使用的面向过程开发方法，被看成是解决软件危机的新兴技术。面向对象技术产生更好的系统结构和更规范的编程风格，极大地优化了数据使用的安全性，提高了程序代码的重用率。面向对象技术确实给编程带来了很大的变化，甚至有些人就此认为用面向对象技术开发出的程序不会存在缺陷，无须进行测试。然而，应该看到，尽管面向对象技术的基本思想保证了软件应该有更高的质量，但实际情况却并非如此，因为无论采用什么样的编程技术，编程人员的错误都是不可避免的，而且由于面向对象技术开发的软件代码重用率高，更需要严格测试，避免错误的繁衍。因此，软件测试并没有因为面向对象编程的

兴起而丧失它的重要性。

从 1982 年在美国北卡罗来纳大学召开首次软件测试的正式技术会议至今,软件测试理论迅速发展,并相应出现了各种软件测试方法,使软件测试技术得到极大的提高。然而,一度实践证明行之有效的软件测试对面向对象技术开发的软件多少显得有些力不从心。尤其是面向对象技术所独有的多态、继承和封装等新特点,产生了传统语言设计所不存在的错误可能性,或者使得传统软件测试中的重点不再显得突出,或者使原来测试经验认为和实践证明的次要方面成为了主要问题。

例如,在传统的面向过程程序中,对于函数 y=Function(x);,只需要考虑一个函数(Function())的缺陷分析;而在面向对象程序中,就不得不同时考虑函数与基类函数(Base∷Function())的行为和继承类函数(Derived∷Function())的行为。

面向对象程序的结构不再是传统的功能模块结构,作为一个整体,原有集成测试所要求的逐步将开发的模块搭建在一起进行测试的方法已成为不可能。而且,面向对象软件抛弃了传统的开发模式,对每个开发阶段都有不同以往的要求和结果,已经不可能用功能细化的观点来检测面向对象分析和设计的结果。因此,传统的测试模型对面向对象软件已经不再适用。针对面向对象软件的开发特点,应该有一种新的测试模型。

面向对象的开发模型突破了传统的瀑布模型,将开发分为面向对象分析(OOA)、面向对象设计(OOD)和面向对象编程(OOP)3 个阶段。分析阶段产生整个问题空间的抽象描述,在此基础上,进一步归纳出适用于面向对象编程语言的类和类结构,最后形成代码。由于面向对象的特点,采用这种开发模型能有效地将分析设计的文本或图表代码化,不断适应用户需求的变动。针对这种开发模型,结合传统的测试步骤的划分,建议采用如图 3.16 所示的测试模型。

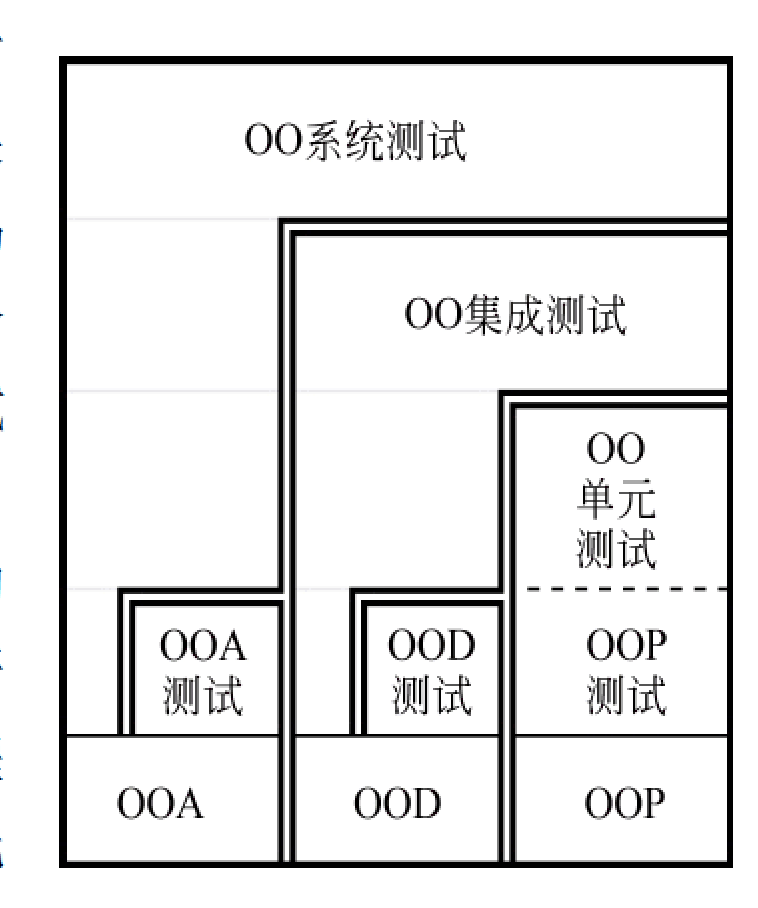


图 3.16 面向对象测试模型

OOA 测试和 OOD 测试是对分析结果和设计结果的测试,主要是对分析设计产生的文本进行测试,是软件开发前期的关键性测试。OOP 测试主要针对编程风格和程序代码实现进行测试,其主要的测试内容在面向对象单元测试和面向对象集成测试中体现。面向对象单元测试是对程序内部具体单一的功能模块的测试,如果程序是用 C++ 语言实现的,主要就是对类成员函数的测试。面向对象单元测试(OO 单元测试)是进行面向对象集成测试的基础。面向对象集成测试(OO 集成测试)主要对系统内部的相互服务进行测试,如成员函数间的相互作用,类间的消息传递等。面向对象集成测试要基于面向对象单元测试。同样,面向对象系统测试要基于面向对象集成测试的最后阶段的测试,主要以用户需求为测试标准,需要借鉴 OOA 或 OOD 测试的结果。

首先看一下面向对象的测试过程。

1. 面向对象分析的测试(OOA Test)

传统的面向过程分析是一个功能分解的过程,是把一个系统看成可以分解的功能的集合。这种传统的功能分解分析法的着眼点在于一个系统需要什么样的信息处理方法和过程,以过程的抽象来对待系统的需要。而面向对象分析(OOA)是“把 E-R 图和语义网络模

型,即信息造型中的概念,与面向对象程序设计语言中的重要概念结合在一起而形成的分析方法”,最后通常是得到问题空间的图表形式的描述。OOA 直接映射问题空间,全面地将问题空间中实现功能的现实抽象化。将问题空间中的实例抽象为对象,用对象的结构反映问题空间的复杂实例和复杂关系,用属性和操作表示实例的特性和行为。对一个系统而言,与传统分析方法产生的结果相反,行为是相对稳定的,结构是相对不稳定的,这更充分反映了现实的特性。OOA 的结果是为后面阶段中类的选定和实现以及类层次结构的组织和实现提供平台。因此,对 OOA 的测试应从以下几个方面考虑。

1) 对认定的对象测试

在 OOA 中,对象是对所描述的问题的抽象。比如 ATM 系统中,客户、管理员、银行和 ATM 机等都是系统所描述的对象。在这里主要从以下几个方面来测试对象:

(1) 认定的对象是否全面,是否问题空间中所有涉及的实例都反映在认定的抽象对象中,是否有遗漏的对象。

(2) 认定的对象是否具有多个属性。只有一个属性的对象通常应看成其他对象的属性,而不将其抽象为独立的对象。

(3) 对认定为同一对象的实例是否有区别于其他实例的共同属性。如果有共同的实例存在,那么如何来处理这种存在重叠的对象。

(4) 对认定为同一对象的实例是否提供或需要进行相同的处理过程,如果不同的实例其处理过程也不相同,认定的对象就需要分解或利用继承性来分类表示。

(5) 如果系统没有必要始终保持对象代表的实例的信息,提供或者得到关于它的服务,认定的对象也无必要。

(6) 认定的对象的名称应该尽量准确、统一。

2) 对认定的结构的测试

认定的结构指的是多种对象的组织方式,用来反映问题空间中的复杂实例和复杂关系。认定的结构分为两种:分类结构和组装结构。分类结构体现了问题空间中实例的一般与特殊的关系,组装结构体现了问题空间中实例整体与局部的关系,如银行与 ATM 系统、账户库、银行储户等。

对认定的分类结构的测试可从如下方面着手:

(1) 对于结构中处于高层的对象,是否在问题空间中含有不同于下一层对象的特殊可能性,即是否能派生出下一层对象。比如对象“汽车”和“轿车”。

(2) 对于结构中处于同一低层的对象,是否能抽象出在现实中有意义的更一般的上层对象。

(3) 对所有认定的对象,是否能在问题空间内向上层抽象出在现实中有意义的对象。

(4) 高层对象的特性是否完全体现下层的共性。

(5) 低层对象是否有高层特性基础上的特殊性。

对认定的组装结构的测试从如下方面入手:

(1) 整体(对象)和部件(对象)的组装关系是否符合现实的关系。

(2) 整体(对象)的部件(对象)是否在考虑的问题空间中有实际应用。

(3) 整体(对象)中是否遗漏了反映在问题空间中有用的部件(对象)。

(4) 部件(对象)是否能够在问题空间中组装新的有现实意义的整体(对象)。

3）对认定的主题的测试

主题是在对象和结构的基础上更高一层的抽象。比如，对于 ATM 取款机，可以将取款、查询等对象抽象成“用户操作”这一主题。对主题层的测试应该考虑以下方面：

（1）遵循 George Miller 的“7＋2”原则，如果主题个数超过 7 个，就要求对有较密切关系的属性和服务的主题进行归并。

（2）主题所反映的一组对象和结构是否具有相同和相近的属性和服务。

（3）认定的主题是否是对象和结构更高层的抽象。

（4）主题间的消息联系（抽象）是否代表了主题所反映的对象和结构之间的所有关联。

4）对定义的属性和实例关联的测试

属性用来描述对象或结构所反映的实例的特性，而实例关联反映实例集合间的映射关系。对属性和实例关联的测试从如下方面考虑：

（1）定义的属性是否对相应的对象和分类结构的每个现实实例都适用。

（2）定义的属性在现实世界是否与这种实例关系密切。

（3）定义的属性在问题空间是否与这种实例关系密切。

（4）定义的属性是否能够不依赖于其他属性被独立理解。

（5）定义的属性在分类结构中的位置是否恰当，低层对象的共有属性是否在上层对象属性中体现。

（6）在问题空间中每个对象的属性是否定义完整。

（7）定义的实例关联是否符合现实。

（8）在问题空间中实例关联是否定义完整，特别需要注意 1-多和多-多的实例关联。

5）对定义的服务和消息关联的测试

定义的服务，就是定义的每一种对象和结构在问题空间所要求的行为。在面向对象编程模式下，更注重的是对象之间的通信、消息的测试。对定义的服务和消息关联的测试从如下方面进行：

（1）对象所描述的服务是否全面。

（2）对象所描述的通信服务是否全面。

（3）消息关联是否正确。

（4）沿着消息关联执行的线程是否合理，是否符合现实过程。

（5）定义的服务是否存在重复。

2．面向对象设计的测试（OOD Test）

在面向对象的设计中，在 OOA 的基础上进行归纳，建立类结构，进一步构造成类库，实现分析结果对问题空间的抽象。由此可见，OOD 是将 OOA 进一步细化。所以，OOD 与 OOA 的界限通常是难以严格区分的。OOD 确定类和类结构不仅是满足当前需求分析的要求，更重要的是通过重新组合或加以适当的补充，能方便地实现功能的重用和扩增，以不断适应用户的要求。因此，对 OOD 的测试应从如下 3 个方面考虑：

（1）对认定的类的测试。OOD 所设计的类是否全面地包含了 OOA 中分析的对象、服务和属性，尽可能减少类之间的依赖性，类中的每一个方法尽可能完成一项服务，或者说是单用途的。

（2）对构造的类结构的测试。在 OOA 中需要分析对象之间的结构关系，那么在 OOD

中当然要对其进行细化。具体体现在：OOD 中应该体现 OOA 中设计的父类和子类的关系，比如，子类是否具有父类没有的新特性；子类间的共同特性是否完全在父类中得以体现。另外，OOD 中应该全面包含 OOA 中的通信和消息关联服务。

(3) 对类库的支持。类库虽然也属于类层次结构的组织问题，但其强调的重点是软件重用。因为它并不直接影响当前软件的开发和功能实现，因此，将其单独提出来测试，也可作为对高质量类层次结构的评估。具体包括：一组子类中关于某种含义相同或基本相同的操作，是否有相同的接口。类中方法(C++ 中为类的成员函数)功能是否较单纯，相应的代码行是否较少(建议不超过 30 行)。类的层次结构是否是深度大、宽度小。

3. 面向对象编程的测试(OOP Test)

面向对象程序具有继承、封装和多态的特性，这使得传统的测试策略必须有所改变。

封装就是对数据的隐藏，外界只能通过程序提供的接口来访问或修改数据，这样降低了数据被任意修改和读写的可能性，降低了传统程序中对数据非法操作的测试。

继承是面向对象程序的一个重要特点，使得代码的重用率提高，同时也使错误传播的概率提高。

多态使得面向对象程序对外呈现出强大的处理能力，但同时却使得程序内“同一”函数的行为复杂化，测试时不得不考虑不同类型具体执行的代码和产生的行为。

面向对象程序是把功能的实现分布在类中，通过类来实现所需要的功能，而需要大量的消息传递来协同完成各个需要的服务。因此，在面向对象编程(OOP)阶段，重点不再是功能实现的细节，而是将测试的目光集中在类功能的实现和相应的面向对象程序风格，主要体现为以下两个方面。

(1) 类是否完成了所要求的功能。

(2) 数据成员是否满足封装的要求。

类所实现的功能都通过类的成员函数执行。在测试时，单独地看待类的成员函数，与面向过程程序中的函数或过程没有任何本质的区别，所以几乎所有传统的单元测试中所使用的方法都可在 OOP 测试中使用。具体的测试方法在前面已经进行了介绍，比如各种黑盒测试方法或者白盒测试方法。

类成员函数间的作用和类之间的服务调用是面向过程测试中很难确定的一个问题。后面章节会提供一些方法以测试类之间通信和调用等服务。

然而，不管采用什么测试方法，OOP 测试必须以 OOD 测试结果为依据，检测类提供的功能是否满足设计的要求，是否有缺陷。必要时(如通过 OOD 测试仍存在不明确的地方)还应该参照 OOA 的结果，以之为最终标准。

4. 面向对象的单元测试(OO 单元测试)

传统的单元测试的对象是软件设计的最小单位——模块。在单元测试中，测试人员需应对模块内所有重要的控制路径设计测试用例，以便发现模块中的错误。同面向过程的测试一样，单元测试采用白盒测试技术，系统内多个模块可以并行地进行测试。然而，在对面向对象的软件进行单元测试时，不同的是，类不能简单地看作是一个模块，类还可以满足继承和多态的要求。

继承的成员函数是否都不需要测试？对父类的测试是否能照搬到子类？

对父类中已经测试过的成员函数,有两种情况需要在子类中重新测试:

(1) 继承的成员函数在子类中做了改动。

(2) 成员函数调用了改动过的成员函数的部分。

当然,父类已经测试过的成员函数,如果在子类中发生改动,对于改动部分还需要设计新的测试用例。

多态有几种不同的形式,如参数多态、包含多态和过载多态。

包含多态和过载多态在面向对象语言中通常体现在子类与父类的继承关系上,对这两种多态的测试可以参考以上的父类和子类的测试。参数多态虽然使成员函数的参数可以有多种类型,但通常只是增加了测试的繁杂。对具有参数多态的成员函数进行测试时,只需要在原有的测试分析和基础上扩大测试用例中输入数据的类型的考虑。

面向对象的继承测试和系统测试将在后续章节进行详细的介绍。

接下来介绍一些常用的面向对象的测试方法,包括有限状态机、Petri 网和正交阵列法等。

3.7.1 有限状态机(FSM)

有限状态机(Finite State Machine,FSM)是一种具有离散输入输出系统的数学模型,它以事件驱动的方式工作,可以通过事件驱动下系统状态间的转移来表达一个控制系统的控制流程。顾名思义,有限状态机里所存在的状态一定是有限个数的,所以有限状态机经常用状态图或者状态迁移图来表示。一般情况下,有限状态机是由输入、状态和输出 3 部分组成的抽象状态集合。

有限状态机理论最早应用在硬件的设计中,后来被逐渐引入软件设计和软件测试领域。有限状态机的状态输出是由当前的输入和过去的输入同时决定的,而输入的结果用"状态"表示。在同一个模型中,一模一样的输入可能会产生不同的输出。也就是说,当前状态决定了模型可能的输入、相应的输出以及由输入产生的状态迁移。有限状态机模型在一个时间点上有且仅有一个活动状态。有限状态机模型可以通过状态转换表或状态转换图表示。有限状态机模型可以用一个四元组来表示$(M,S_0,x,y,\delta,\lambda)$。其中 M 表示有限状态机,S_0 表示有限状态机的初始状态,x 和 y 分别表示输入和输出,δ 和 λ 分别为状态函数和输出函数。

对于一个有限状态机 M,如果对每一个状态和每一个输入都有定义,那么称 M 是完全定义的有限状态机。如果一个给定的有限状态机不是完全定义的,可以通过对未完全定义的状态增加相应的定义来达到完全定义。假如存在未完全定义的有限状态机 M,对每一个未定义的输入 x,定义其指向,如果不存在输出,可以补充一个出错状态或者 NULL 状态,令其指向出错状态或者 NULL 状态。另外,一般情况下,一个完全定义的有限状态机会命名一个 reset 输入,利用 reset 输入可以使状态机中的任意状态回到初态。这样的有限状态机称为具有重置功能的有限状态机。这样,可以使用 reset 分割不同的测试子序列,从而形成一个测试输入序列。如果一个有限状态机 M 可以从初始状态 S_0 到达每一个状态,则称该有限状态机 M 是初始化连通的。如果对于 M 中的每一对状态(S_i,S_j),都存在一个输入序列使得从状态 S_i 能迁移到状态 S_j,则称 M 是强连通的。从上面的定义可知,如果 M 是初始化连通的并具有重置功能,则 M 是强联通的。

目前,根据有限状态和模型完成软件测试,有 4 种方法,分别是 T 方法、D 方法、U 方法

和 W 方法。

(1) T(Transition Tour Method)方法。测试序列要从有限状态机的初始状态 S_0 开始，每一个状态至少执行一次，最终测试要返回到初始状态 S_0。这种方法实际是对有限状态机的全部状态进行遍历，即周游法(Transition Tour Method)，故被命名为 T 方法。这种方法是由时序电路测试方法移植而来的。这种方法最重要的一点是寻找最优测试序列，提高测试效率。

(2) D(Distinguished Sequence)方法。采用 D 方法强调了不同的输出序列(O)。首先对有限状态机每个状态施加相同的输入序列(I)，通过各自不同的输出响应序列来判断有限状态机的当前状态，以此来进行软件检测。这种方法对有限状态机要求较高，需要假定存在最小的、强连通的并且是完备的有限状态机。D 方法的关键是计算有限状态机的区分序列 DS。

(3) U(Unique Input/Output Sequences)方法。测试序列要求每一个状态都有不同的输入输出序列，也就是唯一的输入输出序列(UIO)。首先对每个状态求出所有长度为 1 的输入输出序列；检查它们是否唯一，如果是，那么这个状态的 UIO 序列就找到了；如果不是，对没有 UIO 序列的状态，考虑长度为 2 的输入输出序列；从长度为 k 的输入输出序列中继续求出长度为 $k+1$ 的输入输出序列，检查它们是否唯一，直到对每个状态都找到 UIO 序列。例如，对于如图 3.17 所示的有限状态机，其 UIO 序列及其开销如表 3.23 所示。

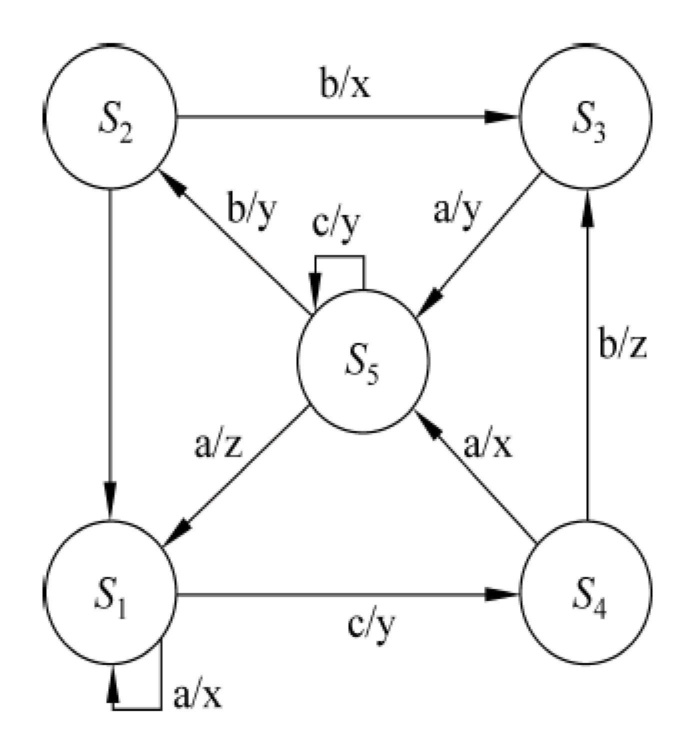


图 3.17 有限状态机描述

表 3.23 UIO 序列及开销

状态	UIO 序列	开销
S_1	a/x, a/x	2
S_2	b/x	1
S_3	a/y, a/z	2
S_4	b/z	1
S_5	a/z	1

(4) W 方法。关键是找到特征序列值 W_set。W_set 可以区分出每一个状态的序列。对于有限状态机 M 中的每一个状态，输入 W_set 中的序列，所得到的最后一位输出均不同。首先构造 M 的特征集 W_set，其次寻找最短路径 SP(一般采用 Dijkstra 算法)，最后结合 W_set 和 SP 生成测试序列。

游戏引擎是有限状态机最为成功的应用领域之一，一个设计良好的有限状态机能够被用来取代部分的人工智能算法，因此游戏中的每个角色或者器件都有可能内嵌一个有限状态机。

例题：考虑 RPG(Role-Playing Game)游戏中城门这样一个简单的对象，它具有打开(Opened)、关闭(Closed)、上锁(Locked)和解锁(Unlocked)4 种状态，如图 3.18 所示。当玩家到达一个处于状态 Locked 的门时，如果此时他已经

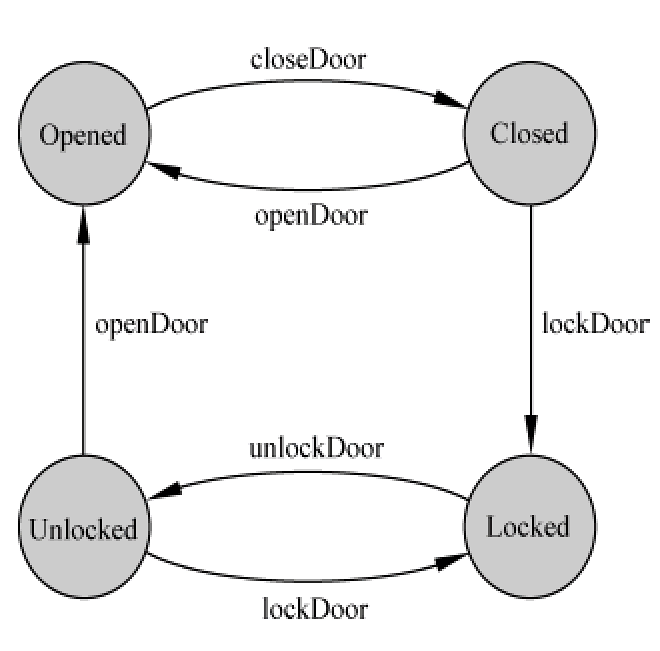


图 3.18 状态表示

找到了用来开门的钥匙，那么他就可以利用它将门的当前状态转变为 Unlocked，进一步还可以通过旋转门上的把手将其状态转变为 Opened，从而成功进入城内，城门关闭，成功进入下一关。

这里假设城门有 4 个状态：S_0：Locked 状态；S_1：Unlocked 状态；S_2：Opened 状态；S_3：Closed 状态。

玩家的输入有 4 个不同的值：A：unlockDoor；B：openDoor；C：closeDoor；D：lockDoor。

玩家的输出有两个不同的值：0：未进入城门，玩家不升级；1：进入城门，玩家升级。

另外设一个输入 R，为 reset 设置，使得状态能够回复到初始状态 S_0。

由于 U 方法和 D 方法对图的要求比较高，这里只采用 T 方法和 W 方法进行测试用例设计。

(1) T 方法：随机地产生输入序列，需要从 S_0 开始，遍历所有的路径，最终回到 S_0。下面的两个不同的测试序列都可以满足 T 方法的要求：

R，A，B，C，D，A，B，C，D，A，D，A，B，C，B，C，D。

R，A，B，C，D，A，B，C，D，A，B，C，B，C，D，A，D。

很明显，这种方法容易产生冗余的测试序列。

(2) W 方法：首先找到 W_set(A，C，D)。

分析：利用输入 A 可以唯一识别 S_0 与其余 3 个状态；利用输入 C，可以唯一识别 S_1、S_2 和 S_3；利用输入 D 可以唯一识别 S_1、S_3。

其次，找到 S_0 到 S_j 的最短路径，并结合 W_set(A，C，D)生成测试序列。

测试序列如下：

R，A，D，A

R，A，B，C，B

R，A，B，C，D

有限状态机能够很好地帮助测试人员发现故障，同时也可以帮助设计人员发现程序中存在不完整的状态定义，并改进软件的设计。然而以上所描述的 4 种测试方法对有限状态机要求比较高，至少需要强连通的有限状态机。所以，目前很多研究人员已经开始想办法对有限状态机进行改进，得到扩展后的 FSM(EFSM)，使其能够满足 U 方法和 D 方法。

3.7.2 Petri 网

Petri 网是在 1962 年由德国科学家 C. A. Petri 博士最早提出来的。它的出现特别适合用来描述离散事件系统的控制流，尤其适合于描述系统中存在的并发特性和异步行为。比如早期 Petri 网用来描述通信网络的协议验证与分析、对分布式数据库系统和实时系统等进行分析。

Petri 网在数学上经常被描述为二元有向图。接下来我们就来了解一下 Petri 网的基本结构。Petri 网主要描述了“条件”和“事件”两大类节点，包括 4 种基本元素：库所、变迁、标识和弧。

一般情况下：“库所”表示“条件”节点，由圆形表示。

“变迁”表示“事件”节点，由矩形或者直线表示，用来描述改变系统状态的事件，例如计

算机和通信系统的信息处理和发送、资源的存取等。直线表示无延时的变迁，而矩形表示有延时的变迁。

弧就是用来连接一个变迁和一个库所或者一个库所和一个变迁的，弧的箭头表示路径的方向。

标识也是Petri网的重要组成部分，它用来描述Petri网的状态信息分布，也称为托肯(token)。它驻留在库所中，控制着变迁的实施。

Petri网着眼于系统中可能发生的各种状态变化以及变化之间的关系，然而系统中状态的变化是通过变迁的实施来完成的。变迁的可实施以及实施规则是Petri网中最重要的规则，它规范了网络中各库所的标识点在变迁发生前后的变化规律，反映了网络状态的变化趋势，使Petri网能够有效地描述和模拟系统的动态特性。

Petri网定义：Petri网是一个双向有向图，记为(P, E, In, Out)，其中P和E为非相交的节点集合，In和Out为边的集合，并有In$\subseteq P\times T$，Out$\subseteq T\times P$。在Petri网中，P表示库所，E表示变迁，In和Out表示边的集合。

1. 顺序执行

如图3.19所示，p1中包含一个标识，事件e1发生，p1中的标识移到p2中，导致事件e2发生，p2中的标识移到p3中，事件e3发生。也就是通过事件e1、e2和e3的顺序发生，状态p1顺序变换到p3。用该Petri网可以模拟一个线性执行过程。库所集合$P=\{$p1,p2,p3$\}$，变迁集合$E=\{$e1,e2,e3$\}$。

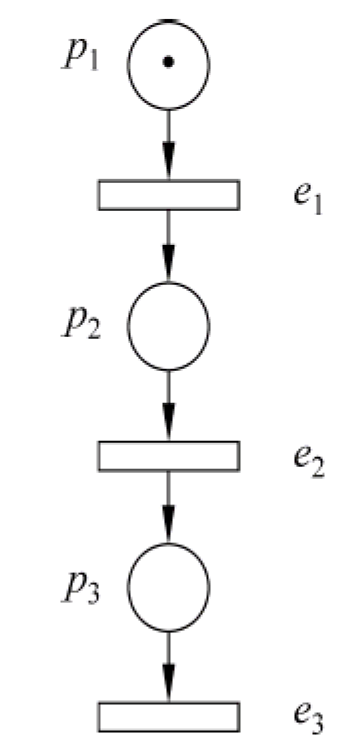


图3.19 Petri网表示顺序执行

2. 同步执行

如图3.20所示，事件e1只有在p1和p2都得到一个标识的时候才能发生，也就是p1和p2执行结束之前不能开始执行p3。假如库所为3种不同的状态，即$P=\{$p1,p2,p3$\}$，那么只有p1和p2状态同时满足才能进入p3状态，这也就实现了同步执行的要求。

3. 并发执行

在Petri网模型中，两个不同的事件可以同时独立地发生，称为并发执行。如图3.21所示，事件e1发生，p1中失去一个标识，库所p2和p3同时各取得一个标识。这时事件e2和e3都可以同时发生，且互不影响。

4. 冲突/选择

如果两个变迁至少共享一个输入库所，则两个变迁在结构上冲突。如图3.22所示，p1中有一个标识，从这个给定的初始条件看，e1和e2都能发生，但不能同时发生，因为它们共享p1中的一个标识，p2和p3中只有一个能取得标识，也就是说事件e1和e2是互相冲突的，冲突反映了系统资源的竞争状况，也就是从p1状态只能到达p2状态，或者到达p3状态。

例3.7 假设有一台自动贩卖机贩售两类商品，一类售价20元，另一类售价50元。该贩卖机只能辨识10元及50元硬币。一开始贩卖机处于Hello的状态，当投入10元时，贩卖机会进入余额不足的状态，直到投入的金额不小于20元为止。如果一次投入50元，则可以选择所有的产品，否则就只能选择20元的产品。当用户完成选择后，贩卖机将会卖出商品并且找回剩余的零钱，随后，贩卖机又将返回初始的状态。

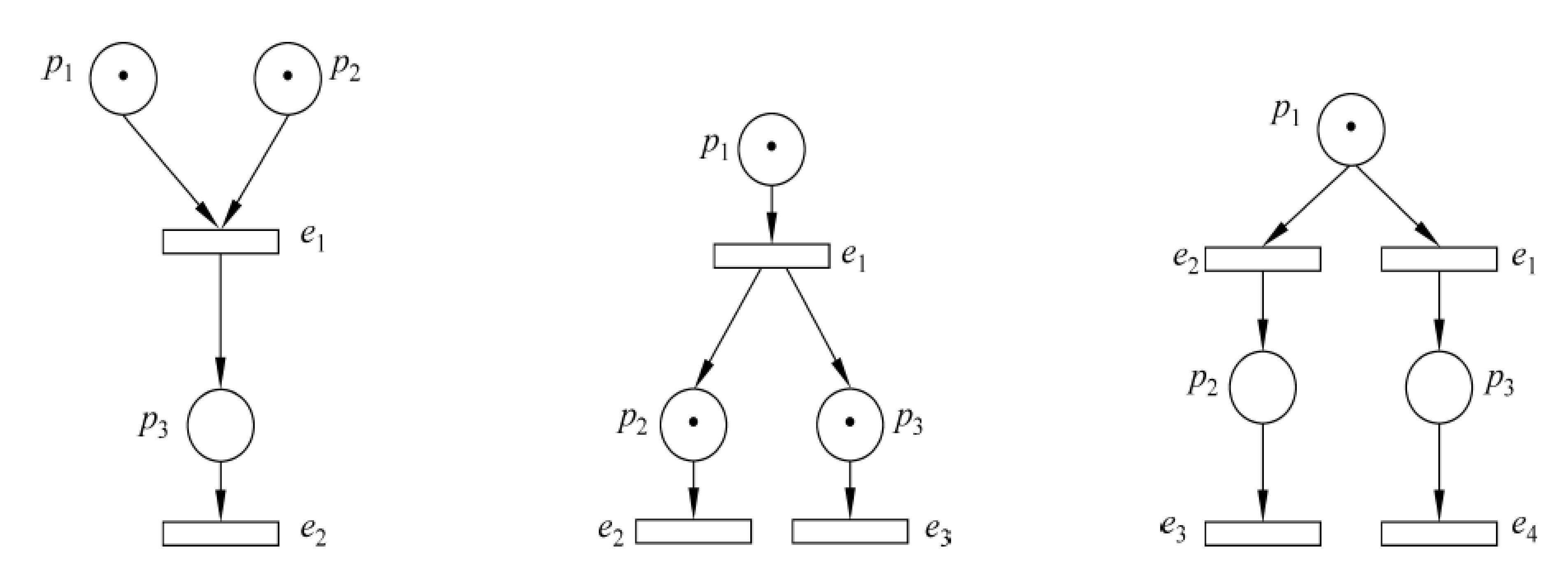


图 3.20 Petri 网表示同步执行　图 3.21 Petri 网表示并发执行　图 3.22 Petri 网表示冲突/选择

图 3.23 中 A～D 表示 4 个不同的库所，其中，A 表示系统处于初始状态(Hello)，B 表示系统将要销售产品，C 表示余额不足，D 表示系统提示销售 20 元商品。

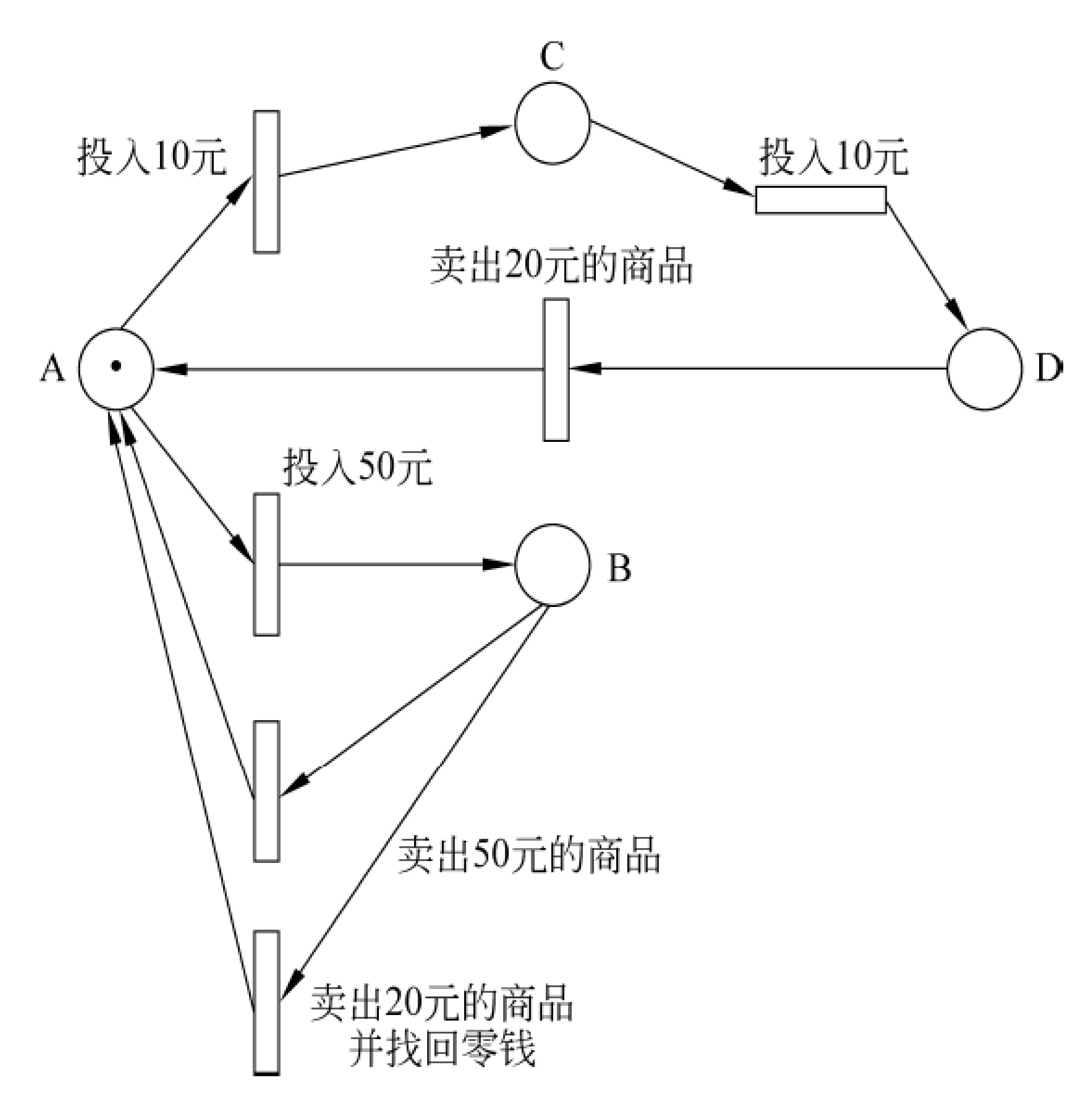


图 3.23 自动贩卖机的 Petri 网表示

3.7.3 正交阵列法

正交阵列测试(Orthogonal Array Testing System，OATS)提供了一种特殊的抽样方法，即通过定义一组交互对象的配对方式的组合，以尽力限制测试配置的组合数目的激增。测试用例的选取就是尽可能找到典型的、覆盖面广的一些数据。正交阵列法是在数理统计和实验设计中经常使用的一种方法。通过这种方法可以找到整齐划一、搭配均衡的实验数据，这正是正交阵列法的特点。

下面举一个简单的例子来看一下正交阵列法的魅力所在。

例 3.8 假设有 A、B、C 三个类，这三个类都需要进行交互操作。程序将这三个类分别进行实例化：A1，A2，A3；B1，B2，B3；C1，C2，C3。那么程序应该产生 $C_3^1 \times C_3^1 \times C_3^1 = 27$ 个交互操作。

为了减少测试数据，同时让测试用例能够涉及 A1，A2，A3；B1，B2，B3；C1，C2，C3 这

9个实例，规定每个实例只出现两次，则只有9种交互操作。这就是正交阵列法。另一项规定是每个实例均出现且只出现两次，也就是要求正交阵列法满足“均衡搭配”的原则；至于“整齐划一”，看一下设计出来的正交表(见表3.24)就明白了。

表3.24 正交表

	A	B	C
1	A1	B1	C3
2	A1	A2	C2
3	A1	B3	C1
4	A2	B1	C2
5	A2	B2	C1
6	A2	B3	C3
7	A3	B1	C1
8	A3	B2	C3
9	A3	B3	C2

采用如表3.24所示的正交表就可以满足A、B、C均衡地进行交互操作，并且每个实例均出现且只出现两次的要求。不但如此，实例还以“整齐划一”的形式出现。表3.24中有9组测试用例，这9种测试用例能够涵盖ABC中A1A2A3、B1B2B3、C1C2C3 9个实例。

正交表是唯一的吗？

接下来看一下正交表的形式化定义。它具体由哪几部分构成，又如何表示这种表格呢？

正交表是一种特别的表格，是正交设计的基本工具。正交表的表示方法为

$$L_N(q^s)$$

其中，

L：正交表；

N：需要做的实验次数(正交表的行数)；

q：各个因素的水平数；

s：正交表的列数(最多能安排的因素个数，包括交互作用、误差等)。

在正交表中经常见到“因素”这个词，因素描述参与正交试验的不同的变量，比如上例中A、B、C就是参与正交试验的3个不同的因素。

这里L和N比较好理解，L就是正交表的名字；N表示正交表的行数。在软件测试中，N也是测试人员最关心的内容，也就是设计的测试用例的数量。

q表示各个因素的水平数，很明显，如果采用这种表示方法，各个因素的水平数是相等的，比如上例中，每个因素均存在3个不同的水平。

当各个因素水平数不同的时候如何表示该正交表，又如何设计正交表呢？

s表示正交表的列数，也就是在满足产生N个不同的实验，且达到“均衡搭配，整齐划一”的原则时所能安排参与试验的最多的因素数，当然希望尽可能多的因素参与到试验中来。

正交表通常按水平数分为3类：

(1) 二水平表，如$L_4(2^3)$，$L_8(2^7)$。

(2) 多水平表，如$L_9(3^4)$，$L_{16}(4^5)$，$L_{25}(5^6)$。

(3) 混合水平表，如$L_{18}(2^1\times3^7)$，$L_{16}(4^2\times2^9)$。

以上例子分别表示什么意思？如何设计其正交表呢？

在这里使用正交阵列法是希望能够改变被测试软件覆盖的完全程度。下面给出一些可能在测试过程中用到的层次。

(1) 穷举性：考虑全部因素的所有可能的组合情况。代价高，信任级高。

(2) 最小性：仅测试每个级别基类之间的交互。测试用例少，信任级低。

(3) 随机性：测试人员随意根据几个类来选择测试用例。信任级不清，测试用例数目任意。

(4) 代表性：统一抽样，确保每个类都被测试到某种程度。对各个类来说信任级相同，测试用例的数目减至最少。

(5) 加权代表性：把用例加入到具有代表性的方法中，以类的相对重要性或类相关联的风险性作为基础。

如何设计正交表呢？现在来回答刚才提出的问题，正交表唯一吗？答案是否定的，只要满足"均衡搭配，整齐划一"就是一个正交表。举一个简单的例子：3因素、2水平的正交表 $L_4(2^3)$ 可设计成表3.25和表3.26的形式。

表3.25　3因素、2水平的正交表 $L_4(2^3)$ 的设计形式(一)

	1	2	3
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1

表3.26　3因素、2水平的正交表 $L_4(2^3)$ 的设计形式(二)

	1	2	3
1	2	1	1
2	1	2	1
3	1	1	2
4	2	2	2

正交表之所以能用较少的试验次数完成原本规模较大的试验，并取得良好的试验效果，正是因为这种方法能够找到最佳或较佳的组合点，也就是满足"均衡搭配，整齐划一"的原则。下面看 $L_4(2^3)$ 正交表。

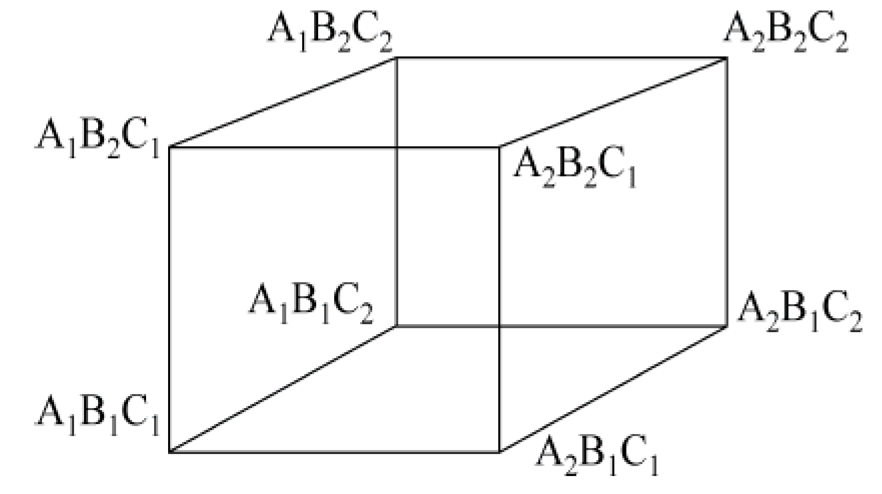


图3.24　正交法说明图

在图3.24中，一个正方体的3个坐标 X、Y、Z 代表了3个因素，每一条边的两个端点代表了两个水平，因此正方体的8个顶点就是全部8次试验，正交设计取了其中的4个点，从图中可以看出来，这4个点的位置恰好使得正方体的每个面上都有两个点，每条边上都有一个点；因此，这4个点是均衡地分布在这一正方体上的，这就是"均衡搭配"的原则；由于试验点分散得很均衡，因此其代表性就很强，能很全面地反映试验的最佳或较佳结果。从正交表的每一个列来看，各个水平都以"整齐划一"的方式出现。

那么如何来设计正交表呢？如果是简单的正交表，可以根据以上内容自行来设计；如果是复杂的正交表，可以参考如下资源来设计：

✍ http://www.support.sas.com/techsup/technote/ts723_Designs.txt

✍ http://www.york.ac.uk/depts/maths/tables/orthogonal.htm

✍ 数理统计实验设计等方面的书

如果某些因素对实验的结果影响特别大，或者由于某种目的，必须考虑某些因素的效应，则可以增加部分因素的水平数，此时可以考虑选用混合水平正交表。

在混合水平正交表中，各个因素的水平数不同，那么如何来设计正交表呢？

(1) 首先根据水平数去选。

(2) 其次根据因素个数去选。一般，因素的个数可以少于正交表的列数，在使用时将多余的列划掉。

注意：如果需要考虑交互作用，则将一组交互作用当作一个单独的因素，在选表时因素的个数应该包括交互作用的个数。

例 3.9 图 3.25 是在线购物系统的“购物车”页面的一部分，这部分的目的是计算产品价格，需要统计用户信息。

○ 会员 ○ 普通用户

○ 是否使用礼品券
○ 是否使用优惠券

图 3.25 在线购物系统的“购物车”页面的一部分

分析：测试内容中包含 3 个因素：

(1) 用户是否为该购物网站会员。

(2) 用户是否使用礼品券。

(3) 用户是否使用优惠券。

如果要全面测试这个部分，需要将上述 3 个因素进行 2×2×2 组实验数据完成测试过程。即：

(1) 会员使用礼品券和优惠券。

(2) 非会员使用礼品券和优惠券。

(3) 会员使用优惠券。

(4) 非会员使用优惠券。

(5) 会员使用礼品券。

(6) 非会员使用礼品券。

(7) 会员不使用礼品券和优惠券。

(8) 非会员不使用礼品券和优惠券。

使用正交阵列法分析：该问题符合 $L_4(2^3)$ 正交表。设计的正交表如表 3.27 所示。

表 3.27 在线购物系统的正交表设计

因素 次数	会员	礼品券	优惠券
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0

说明：0 表示否定，1 表示肯定。

因此设计 4 个不同的测试用例，分别为：(1)非会员不使用任何礼品券和优惠券；(2)非会员使用礼品券和优惠券；(3)会员使用优惠券；(4)会员使用礼品券。

3.7.4 UML 软件测试

统一建模语言 UML 是面向对象技术的重要内容，如今已成为软件开发过程中不可缺少的关键性环节。UML 是对软件开发过程进行可视化的一种表示方式，是一种统一的标准语言，它融合了多种方法的成果，定义良好，功能强大，普遍适用，适合描述面向对象技术

开发的所有软件和应用程序等，具有很宽的应用领域；而且 UML 适用于系统开发从需求规格描述直至系统维护的不同阶段。

UML 提供了 9 种图，包括类图、对象图、用例图、交互图(包括顺序图和合作图)、状态图、活动图、组件图、配置图。UML 通过 4 层元模型(metamodel)体系结构来定义。分别是元元模型、元模型、模型和用户对象。元元模型定义元模型，元模型定义说明模型，模型描述具体信息领域；用户对象是模型的一个实例。除此之外，UML 还被分成 3 个逻辑子包：基础设施包、行为元素包和模型管理包。

基础设施包给出了系统静态结构的基本架构，包括类图、对象图、组件图和配置图。

行为元素包用于支持系统动态行为建模，包括用例图、交互图(包括顺序图和合作图)、活动图和状态图。

1. UML 用例图测试

在传统的软件开发方法和早期的面向对象的开发方法中，都是用自然语言来描述系统的功能需求。这样的做法没有一个统一的格式，缺乏形式化的描述，随意性较大，容易产生理解上的混淆和不准确的表示。而 UML 中的用例图(Use Case Diagram)应运而生，很好地解决了这个问题。

用例图是建立需求获取的模型。用例模型用于需求分析阶段，从用户的角度来描述系统的功能。用例图包括三大元素：用例、参与者以及二者之间的通讯关联。

用例是外部可见的一个系统功能单元，它可以与一个或多个参与者进行信息传递和交互。用例中必须包含它所拥有的所有的行为。

参与者是与系统、子系统或类发生交互的外部用户、进程或其他系统；参与者可以是人，也可以是另一台计算机或者进程。在实际运作中，一个实际用户可能对应系统的多个参与者，而不同的用户也可能对应同一个参与者。每个参与者都可以与一个或多个用例进行交互。

下面给出一个 ATM 系统的简单的用例表示，如图 3.26 所示。

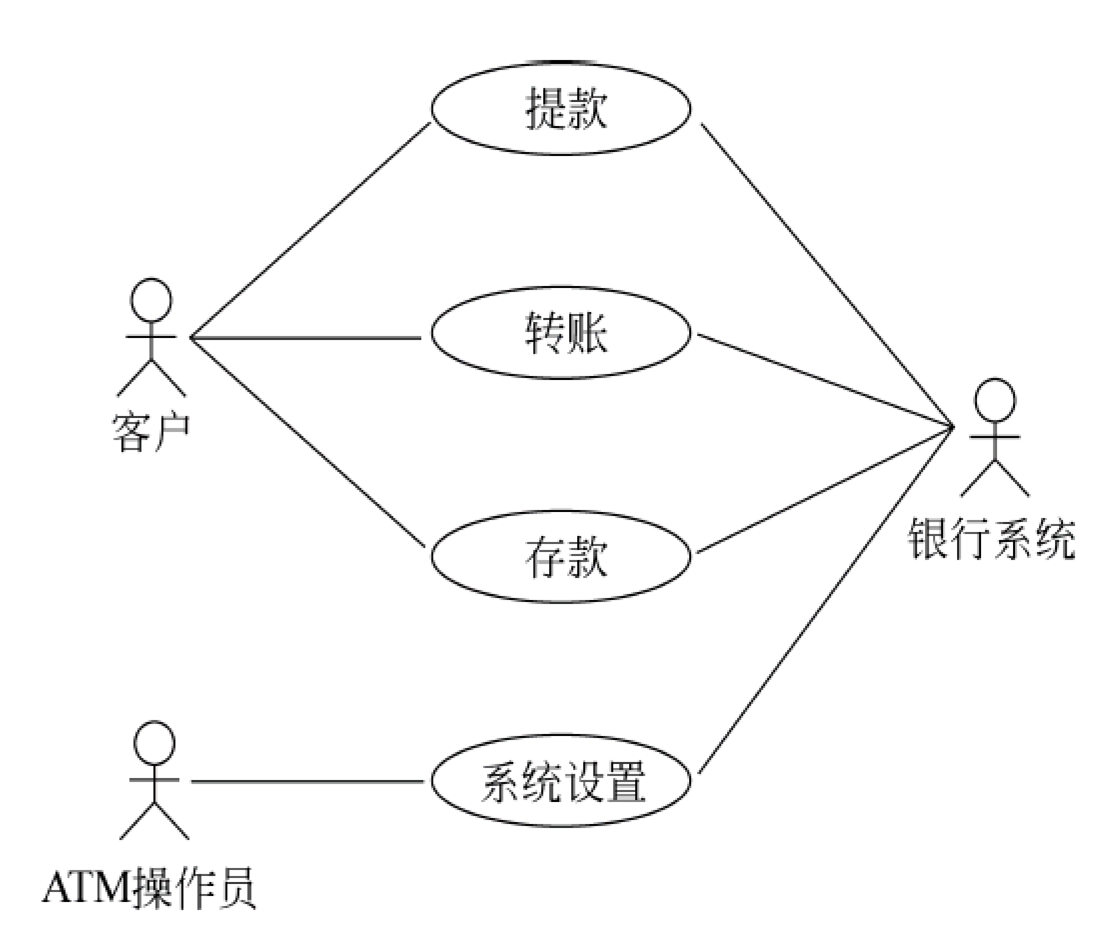


图 3.26 ATM 系统用例图

图中有 4 个用例，分别为提款、转账、存款以及系统设置。参与者为客户、ATM 操作员和银行系统。客户可以参与提款、转账和存款等用例过程；银行系统可以参与提款、转账、存款和系统设置等用例；ATM 操作员可以参与系统设置用例。

1）用例图的测试

用例图经常会转换成若干场景，采用场景法来进行测试。场景由一系列的相关活动组成，能够表示行为的一个特定的动作序列，也可以称之为事件序列，因此场景法可以构成若干个不同的事件流。它就像一个剧本，可以用来演绎系统未来预期的使用过程。场景可以看作是站在用户的视角来描述用户与系统的交互，而定义了必须实现的软件功能的功能需求说明则可以看作是用户需求分解的结果。设置场景的目的是让所有人员明白用户的目标是什么，以及用户希望怎样做，而这一过程并不涉及界面展现、程序实现等细节的内容。

场景描述是一个迭代细化的过程，需要有清晰、明确的上下文环境。通过与系统的交互图相结合，来共同描述与系统的交互行为。场景是用例的一个实例。一个简单的场景是通过一个用例定义一些相关的数据以及覆盖这个用例所流经的路径。定义的数据通过输入、输出以及一些中间状态与具体的场景相关联。一个复杂的场景表明了多个功能之间的信息流是如何进行运作的，它可能包含了多个用例，通过控制场景或子场景的执行顺序、条件控制、并行或反复处理组合而成。

简单地说，每个事件触发时的情景就是一个场景，而同一事件不同的触发顺序和处理结果形成事件流。

一个场景包括基本流和备选流，经过遍历所有的基本流和备选流来完成整个场景。

基本流：一般采用直线表示，经过用例的最简单的路径（不出现任何差错，程序从开始直接执行到结束）。

备选流：用曲线表示，一个备选流可能从基本流开始，在某个特定的条件下执行，然后重新加入基本流中；也可能起源于另一个备选流，结束用例，而不加入基本流。

用场景法描述事件流如图3.27所示。

设计场景如下：

场景1：基本流；

场景2：基本流，备选流1；

场景3：基本流，备选流1，备选流2；

场景4：基本流，备选流3；

场景5：基本流，备选流3，备选流1；

场景6：基本流，备选流3，备选流1，备选流2；

场景7：基本流，备选流4；

场景8：基本流，备选流3，备选流4。

图3.27 场景法描述事件流图

2）场景法测试设计

在进行场景法测试设计时，首先根据用户规格说明书，将一个用例或者多个用例的组合描述为基本流和备选流，根据基本流和备选流生成不同的场景，要求场景覆盖到基本流和全部的备选流。测试人员要为每一个场景生成不同的测试用例。之后，根据需要对测试用例进行审查，去掉多余的测试用例，确定测试用例后，为每一个测试用例确定测试数据值。

一般情况下，一个业务只存在一个基本流，一个基本流只有一个起始点和一个终止点。如果某一个业务出现两个基本流，可将其看做不同的业务。备选流可以起源于基本流，也可以起源于其他备选流；备选流结束于一个流程出口，或者汇入基本流或其他备选流。

3）ATM 的场景图

接下来解决 ATM 的场景图和测试用例等问题。

根据 ATM 用例图，以“提款”业务为例分析场景，其中 ATM 系统的场景图如图 3.28 所示。其业务流程如下。

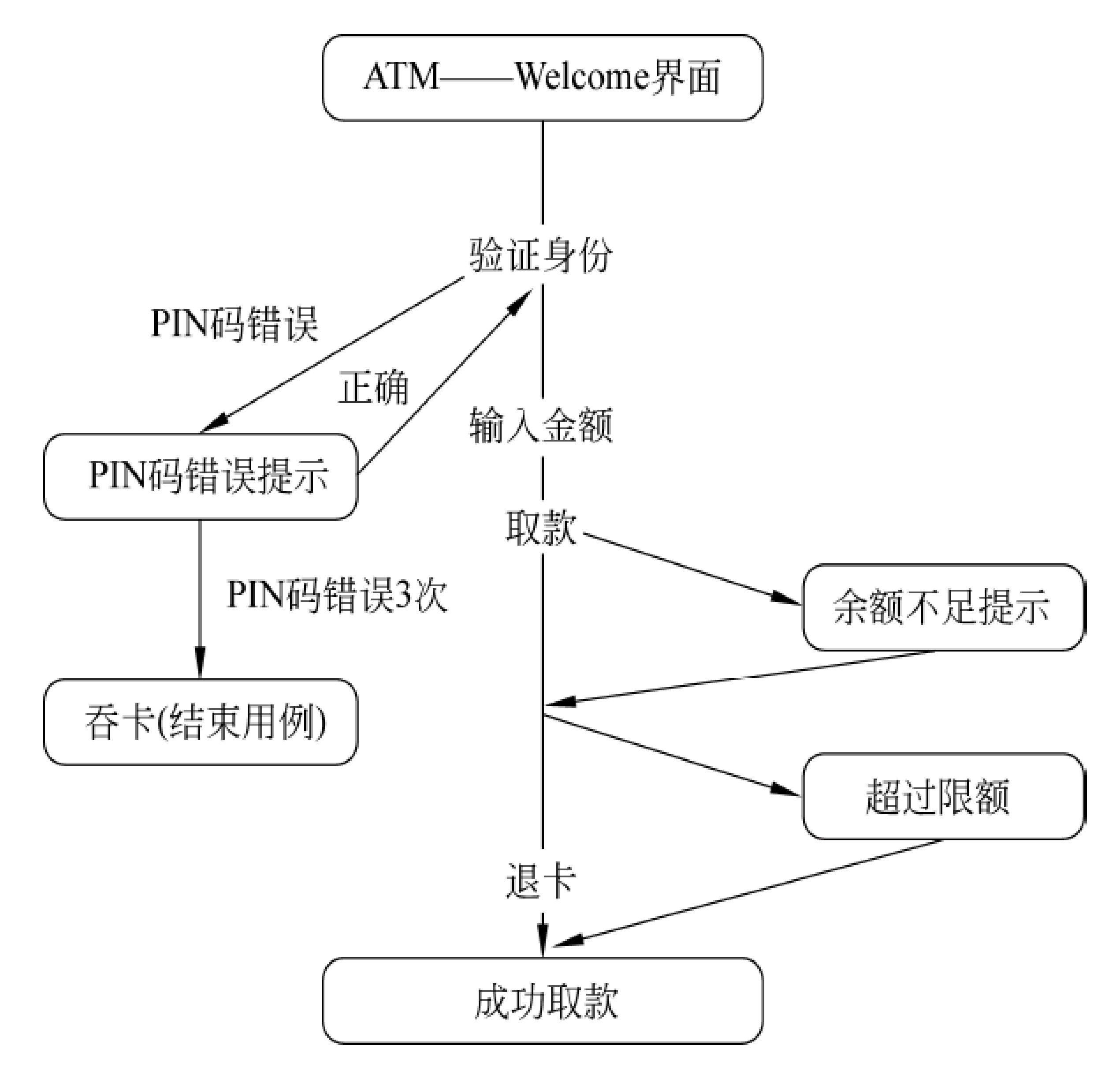


图 3.28　ATM 系统场景图

基本流：成功提款 n 元；

备选流 1：密码不对；

备选流 2：密码输入错误 3 次；

备选流 3：余额不足；

备选流 4：取款额超过 24 小时限额。

设计场景用例如表 3.28 所示。

表 3.28　ATM 系统的场景用例设计表

	参与者	用例	验证身份	取款金额	结　束
场景 1	客户	取款	PIN 码正确	500	成功取款
场景 2	客户	取款	PIN 码错误 1 次	500	成功取款
场景 3	客户	取款	PIN 码错误 3 次	500	吞卡
场景 4	客户	取款	PIN 码正确	2500	余额不足
场景 4	客户	取款	PIN 码正确	3100	取款额超过 24 小时限额

2. 交互图测试

面向对象设计方法中，对象之间通过交互来实现各种功能。对象的交互是面向对象的一大特点。所以 UML 语言对于对象交互过程从不同侧面给出了不同的描述形式。其中主要有时序图(sequence diagram)和协作图(collaboration diagram)两种。时序图主要关注交

互的时序关系,而协作图主要描述对象结构信息以及对象在交互中的关系等。

首先看一下 UML 状态图的测试。UML 状态图可以参考有限状态机的内容,属于状态变化的测试。

图 3.29 是手机状态图,当手机开机时处于 idle 状态;用户使用手机呼叫某人时手机进入 dialing 状态,呼叫成功进入 working 状态;当有人呼叫本机时进入 ringing 状态。

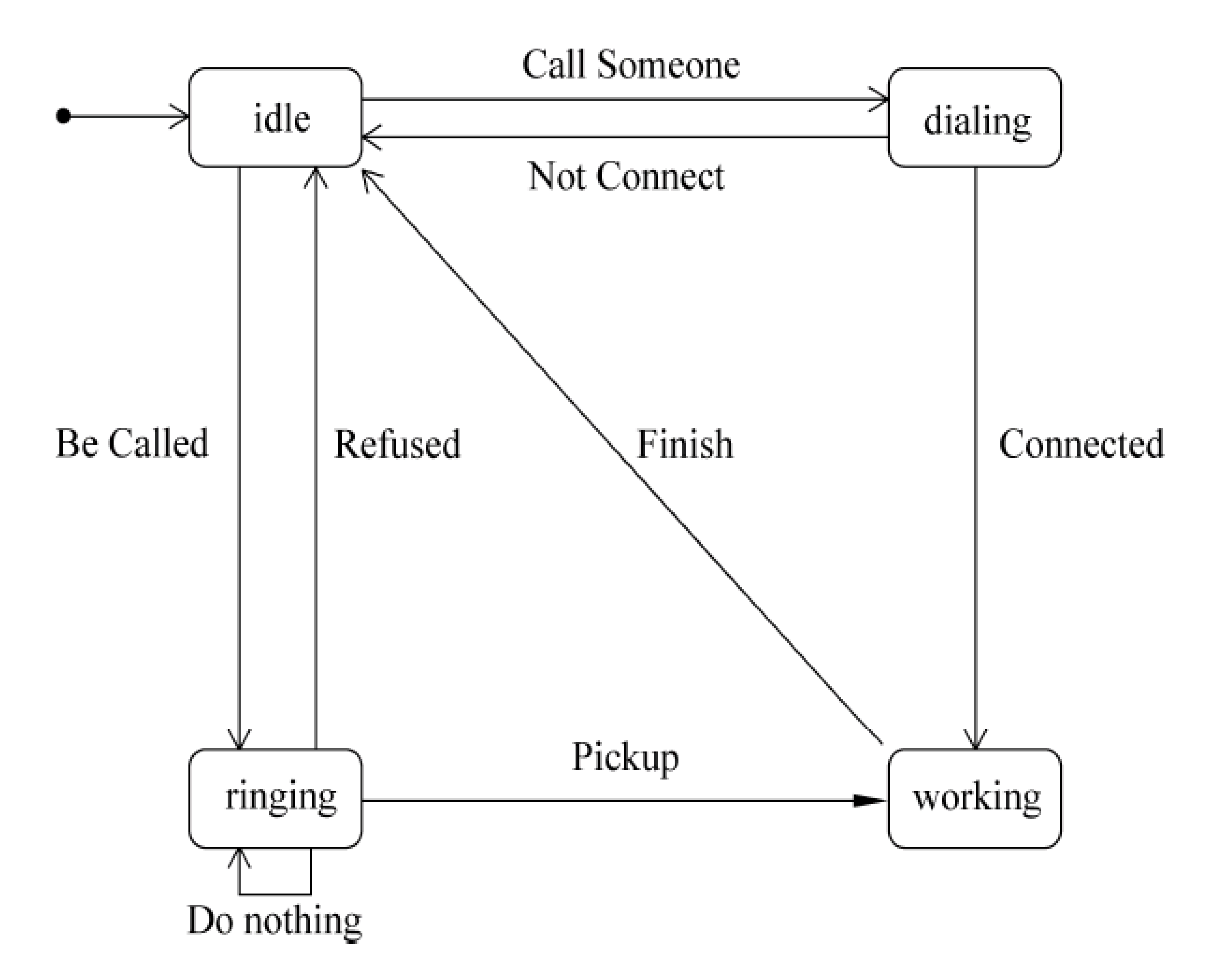


图 3.29 手机接打电话状态图

采用 T 方法(周游法)设计测试用例:

(1) idle 状态利用 Call Someone 事件到达 dialing,利用 Not Connect 事件到达 idle。

(2) idle 状态利用 Call Someone 事件到达 dialing,利用 Connected 事件达到 working,利用 Finish 事件到达 idle 状态。

(3) idle 利用 Be Called 事件到达 ringing,利用 Pickup 事件到达 working,利用 Finish 事件到达 idle。

(4) idle 状态利用 Be Called 事件到达 ringing 状态,利用 Refused 事件到达 idle。

对于时序图来说,时序图已经提供了很好的测试顺序,按照时序图中,对象发生的先后顺序进行测试即可。自动饮料售货机的时序图如图 3.30 所示。

(1) 顾客从投币口塞入现金,然后选择想要的饮料;

(2) 系统将钱币送入现金记录仪;

(3) 记录仪检查是否还有存货;

(4) 记录仪更新自己的现金存储记录;

(5) 记录仪通知饮料分配器传送一罐饮料到出货口。

测试用例按照时序图的时序进行测试。设计用例如下:

(1) 顾客购买商品,没有零钱,返还现金。

(2) 顾客购买饮品,无需找零,库存充足。

(3) 顾客购买饮品,需找零 5 元,库存充足。

(4) 顾客购买饮品,无需找零,库存不足,返还现金。

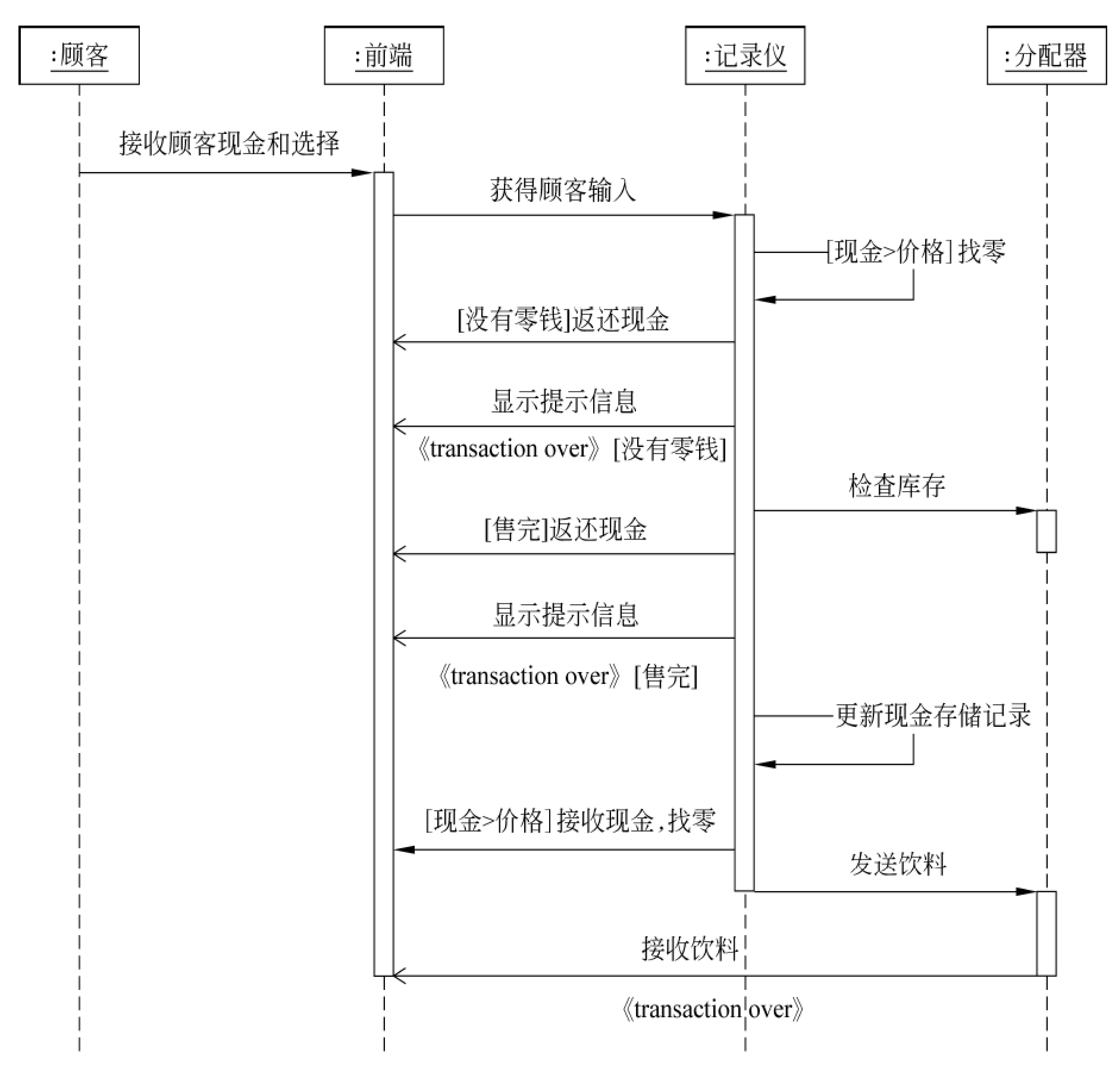


图 3.30　自动饮料售货机的时序图

3.7.5　案例分析——UML 描述的 ATM 系统软件测试用例设计

ATM 是由计算机控制的持卡人自我服务型的金融专用设备。在 ATM 上,用户也可以进行账户查询、修改密码和转账的业务。ATM 系统向用户提供了一个方便、简单、及时存取款的现代计算机化的网络系统。可以大大减少工作人员,节约人力资源的开销,同时由于手续简便,也可以减轻业务员的工作负担,有效地提高整体的工作效率和精确度。

ATM 系统,是一个由终端机、ATM 系统和数据库组成的应用系统。系统功能有用户在 ATM 上提取现金、查询账户余额、修改密码及转账功能。然而作为自助式金融服务终端,ATM 除了提供金融业务功能之外,还具有维护、测试、事件报告、监控和管理等多种功能。

1. 需求分析

ATM 系统是一个复杂的软件控制硬件系统。首先进行系统总体功能需求分析。一个完整的 ATM 机应该包括以下几个模块。

- 读卡机模块：客户通过读卡机模块送入银行卡,ATM 可以识别出不同银行的银行卡和客户信息。
- 键盘模块：客户使用该模块可以输入密码、取款金额或者选择要进行的操作。
- IC 认证模块：安全授权系统,辨别真伪。

- 显示模块：显示客户信息，方便与客户交互。
- 吐钞机模块：通过该模块，客户可以取钱。
- 打印表模块：提供给客户取款凭据。
- 监视器模块：视频监视，记录视频信息。

根据需求分析建立用户用例图。

客户用例图如图 3.31 所示。

客户可以发起“存钱”“取钱”“转账”“改密码”和“查余额”等用例。

ATM 管理人员用例图如图 3.32 所示。

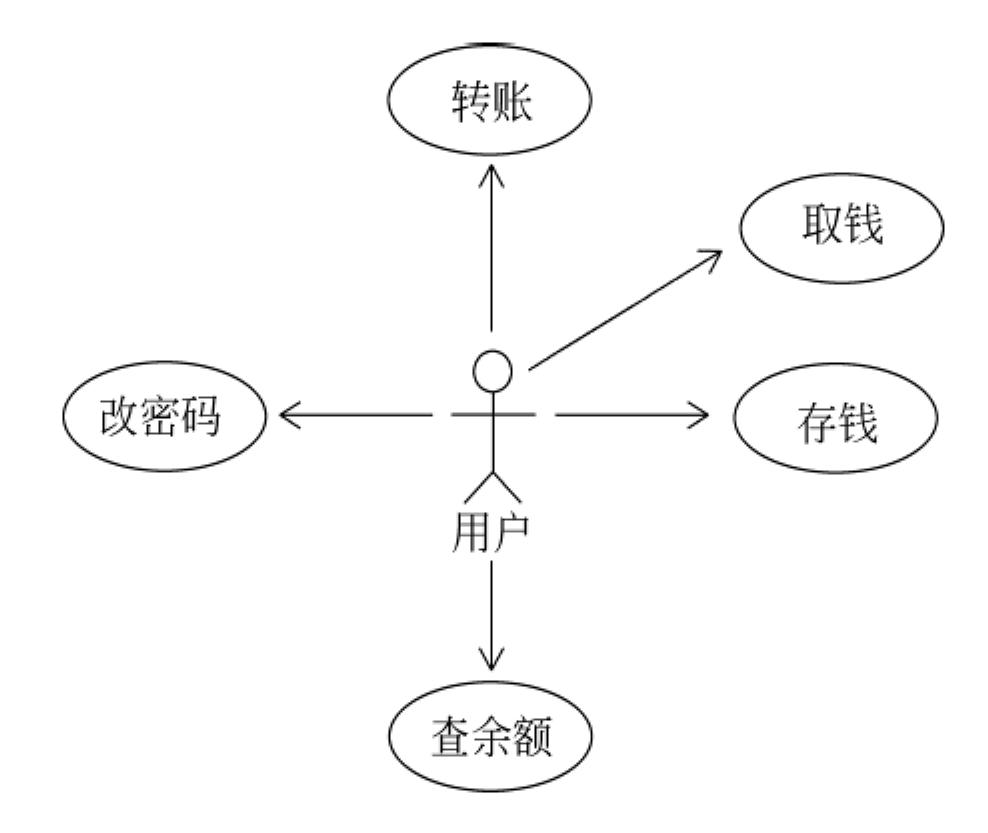


图 3.31　ATM 系统中的客户用例关系图

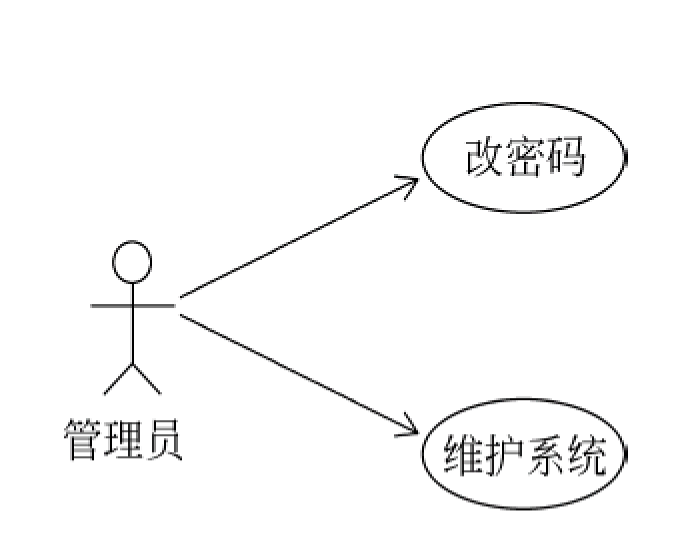


图 3.32　ATM 管理人员用例图

根据用例图，分析用户的事件流如下。

1）取钱事件流

（1）基本流

① 用户输入取款金额。

② 系统验证输入金额符合输入要求。

③ 系统显示用户取款金额。

④ 用户确认取款金额。

⑤ 系统要求吐钞机出钞。

⑥ 系统更新并保存账户信息。

（2）备选流

① 用户输入密码不正确，重新输入。

② 用户输入密码错误 3 次，吞卡。

③ 用户卡中余额不足，退出。

④ 用户取款额超过 24 小时限额，退出。

2）存钱事件流

（1）基本流

① 用户放入现金。

② 系统验证现金。

③ 系统显示金额。

④ 用户确认金额。

⑤ 系统更新用户账户信息并保存。
⑥ 用户存款成功。
(2) 备选流
① 系统识别钱币错误,退钱。
② 用户不确认,给出提示,退出。
3) 改密码事件流
(1) 基本流
① 用户输入旧密码。
② 系统验证账户旧密码。
③ 用户输入两次新密码。
④ 用户确认输入的密码。
⑤ 系统更新用户密码为新密码。
⑥ 用户修改密码成功。
(2) 备选流
① 如果输入的旧密码错误,给出提示,退出。
② 如果两次输入的密码不同,给出提示,退出。
③ 如果用户没有确认,给出提示,退出。
4) 转账事件流
(1) 基本流
① 用户输入转账账号。
② 系统验证转账账号。
③ 用户输入转账金额。
④ 系统验证输入金额是否符合输入要求。
⑤ 系统验证用户账户余额。
⑥ 系统显示用户转账账户及转账金额。
⑦ 用户确认转账账户及转账金额。
⑧ 系统更新并保持账户信息。
(2) 备选流
① 如果输入账号不正确,给出提示,退出。
② 如果输入金额不符合输入格式要求,给出提示,退出。
③ 如果输入金额超出最大转账金额,给出提示,退出。
④ 如果用户没有确认,给出提示,退出。

2. 系统动态模型

这里以时序图为例设计测试用例,ATM 系统的时序图如图 3.33 所示。

测试用例按照时序图的时序进行测试。设计用例如下:

(1) 客户修改密码,磁卡无效。
(2) 客户修改密码,PIN 码验证不成功。
(3) 客户修改密码,修改成功。

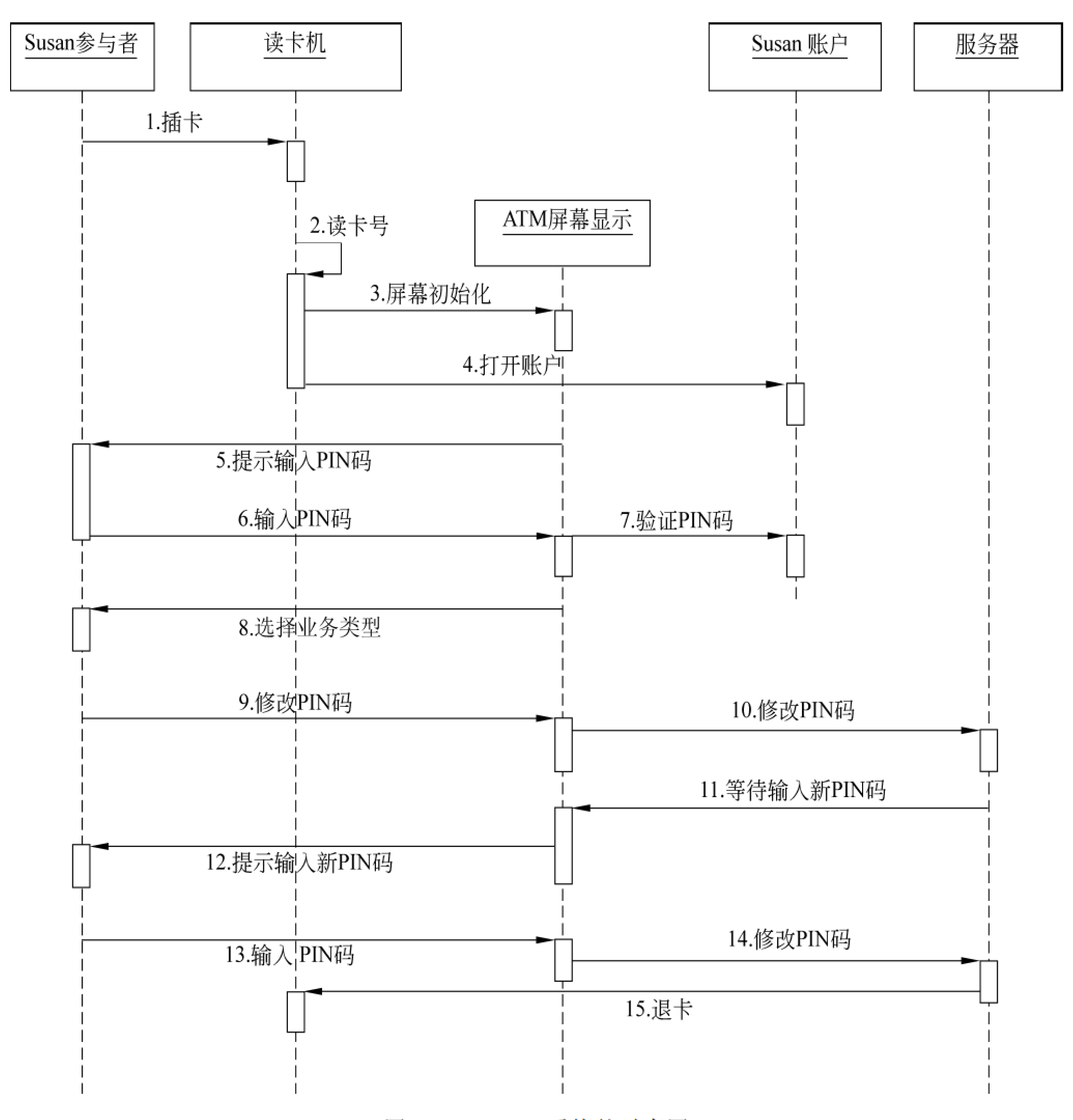


图 3.33　ATM 系统的时序图

3.8　本章小结

本章主要介绍了黑盒测试方法，白盒测试方法及面向对象软件测试技术。首先介绍了黑盒测试方法，包括边界值分析法，等价类划分法，因果图法及决策表法；其次介绍了白盒测试方法，包括基于路径的测试、数据流测试方法及逻辑覆盖准则；最后介绍了面向对象的测试方法，包括正交试验法、Petri 网和有限状态机等测试方法。本章结合大量的简单易懂的案例来介绍相关的软件测试方法，帮助读者更好地学习软件测试相关的理论。

习　　题

1. 给出某一天(年、月、日)，计算出它的下一天，取值范围为

年：1000～3000

月：1～12

日：1～31

如 2013 年 3 月 4 日的下一天是 2013 年 3 月 5 日。

要求如下。

输入：3 个参数(年、月、日)

输出：如能正确计算，输出它的下一天，否则输出相应的错误信息。

请结合等价类划分法和边界值分析法设计出相应的测试用例。

2. 对一个自动饮料售货机软件进行黑盒测试设计。该软件的规格说明如下：“有一个处理单价为 1 元 5 角的盒装饮料的自动售货机软件。若投入 1 元 5 角硬币，按下“可乐”、“雪碧”或“红茶”按钮，相应的饮料就送出来。若投入的是 2 元硬币，在送出饮料的同时退还 5 角硬币。”

(1) 试利用因果图法建立该软件的因果图。

(2) 设计测试该软件的全部测试用例。

3. 白盒测试的测试方法有哪些?

4. 使用基本路径测试方法，为以下程序设计测试用例。

```
int IsLeap(int year){
    if(year % 4 ==0){
        if(year % 100 ==0){
            if(year % 400 !=0)
                leap=1;
            else
                leap=0;
        }else
            leap=1;
    }else
        leap =0;
    return leap;
}
```

(1) 画出控制流图。

(2) 计算环形复杂度。

(3) 列出基本路径并设计测试用例。

5. 根据以下程序，分别完成语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖和条件组合覆盖测试用例的设计。

```
void DoWork(int x,int y,int z){
    int k=0,j=0;
    if((x>3)&&(z<10)){
        k=x* y-1;
        j=sqrt(k);
    }
    if((x==4)||(y>5)){
```

```
        j=x* y+10;
    }
    j=j%3;
}
```

6. 图 3.34 是图书管理系统借阅者用例图，分析"借书""还书"的场景并画出事件流图，设计测试用例。

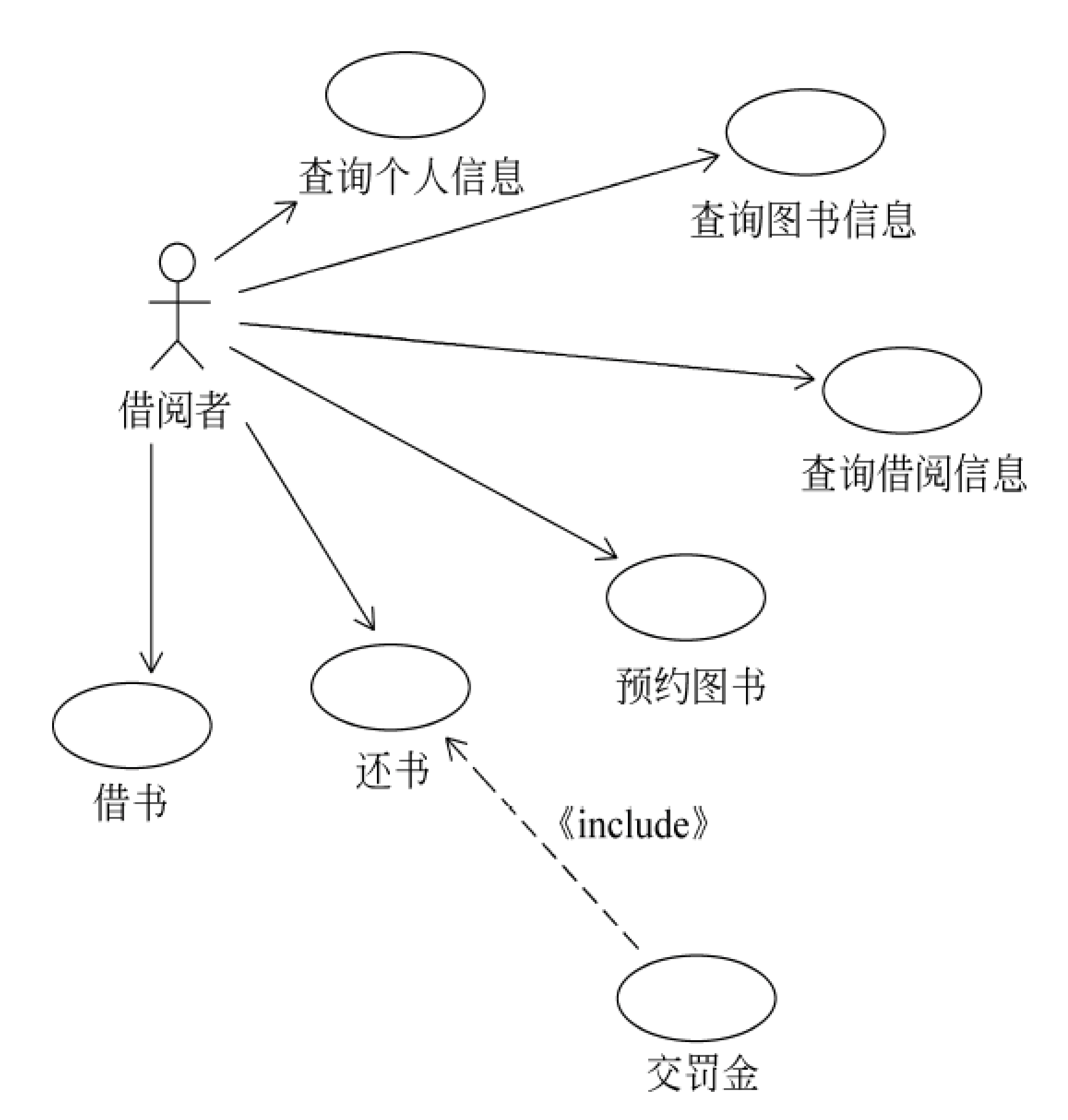


图 3.34　借阅者用例图

第4章　集成测试

集成测试中非常著名的一个案例是1999年美国宇航局的火星基地登陆飞船在试图登陆火星表面时突然坠毁失踪。从理论上看，登陆计划是这样的：在飞船降落到火星的过程中，降落伞将被打开，减缓飞船的下落速度。降落伞打开后的几秒钟内，飞船的3条腿将迅速撑开，并在预定地点着陆。当飞船离地面1800m时，它将丢弃降落伞，点燃登陆推进器，在余下的高度缓缓降落地面。美国宇航局为了省钱，简化了确定何时关闭推进器的装置。为了替代其他太空船上使用的贵重雷达，在飞船的脚上装了一个廉价的触点开关，在计算机中设置一个数据位来关掉燃料。其原理很简单，飞船的脚不着地，引擎就会点火。不幸的是，质量管理小组在事后的测试中发现，当飞船的脚迅速摆开准备着陆时，机械震动在大多数情况下也会触发着地开关，设置错误的数据位。设想飞船开始着陆时，计算机极有可能关闭推进器，而火星登陆飞船下坠1800m之后冲向地面，必然会撞成碎片。

事后调查发现，仅仅由于两个测试小组单独进行测试，没有进行很好的沟通，缺少一个集成测试的阶段，结果导致事故的发生。质量管理小组观测到故障，并认定出现误动作的原因极可能是某一个数据位被意外更改。什么情况下这个数据位被修改了？又为什么没有在内部测试时发现呢？为什么会出现这样的结果？经过分析发现，登陆飞船经过了多个小组测试。其中一个小组测试飞船的脚落地过程(leg fold-down procedure)，但从没有检查那个关键的数据位，因为那不是这个小组负责的范围；另一个小组测试着陆过程的其他部分，但这个小组总是在开始测试之前重置计算机，清除数据位。双方本身的工作都没什么问题，就是没有合在一起测试，其接口没有被测试，而问题就在这里，后一个小组没有注意到数据位已经被错误设定。就是由于这样一个集成测试问题，导致本次事故造成了重大的损失。

4.1　集成测试概念

4.1.1　集成测试简介

集成测试，也叫组装测试或联合测试，是在单元测试的基础上，将所有模块按照设计要求(如根据结构图)组装成为子系统或系统进行测试。实践表明，一些模块虽然能够单独地工作，但并不能保证连接起来也能正常工作。程序在某些局部反映不出来的问题，在全局上很可能暴露出来，影响功能的实现。

所有的软件项目都不能摆脱系统集成这个阶段。不管采用什么开发模式，具体的开发工作总得从一个一个的软件单元做起，软件单元只有经过集成才能形成一个有机的整体。具体的集成过程可能是显性的，也可能是隐性的。只要有集成，总是会出现一些常见问题，工程实践中几乎不存在软件单元组装过程中不出任何问题的情况。从图4.1可以看出，集成测试需要花费的时间远远超过单元测试，直接从单元测试过渡到系统测试是极不妥当的做法。

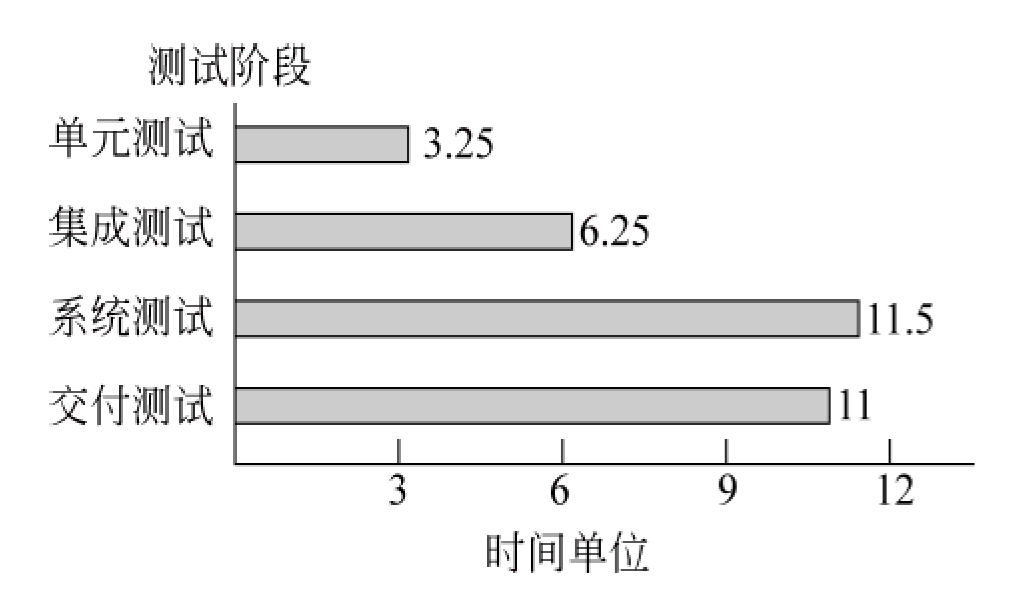


图 4.1　针对一个功能点的各类测试所花费的时间统计

4.1.2　集成测试的目的和意义

在集成测试中需要考虑以下问题：

(1) 在把各个模块连接起来的时候，通过模块接口的数据是否会丢失。

(2) 各个子功能组合起来，能否达到预期要求的父功能。

(3) 一个模块的功能是否会对另一个模块的功能产生不利的影响。

(4) 全局数据结构是否有问题。

(5) 单个模块的误差积累起来，是否会放大，从而达到不可接受的程度。

要想发现并排除在模块连接中可能发生的上述问题，就需要进行集成测试。

集成测试具有其他测试不可替代的特点：

(1) 单元测试具有不彻底性，对于模块间接口信息内容的正确性、相互调用关系是否符合设计无能为力。只能靠集成测试来进行保障。

(2) 同系统测试相比，由于集成测试用例是从程序结构出发的，目的性、针对性更强，测试项发现问题的效率更高，定位问题的效率也较高。

(3) 能够较容易地测试到系统测试用例难以模拟的特殊异常流程，从纯理论的角度来讲，集成测试能够模拟所有实际情况。

(4) 定位问题较快，由于集成测试具有可重复强、对测试人员透明的特点，发现问题后容易定位，所以能够有效地加快进度，减少隐患。

4.2　集成测试方法

集成测试的策略比较多，如有基于功能分解的集成，基于调用图的集成、基于路径的集成、分层集成、高频集成、基于进度的集成、基于风险的集成和基于使用的集成等。一般的软件测试及软件工程中按照功能分解将集成测试方法分为非渐增式集成(大爆炸集成)、渐增式集成和三明治集成。

4.2.1　非渐增式集成测试

非渐增式集成也称为大爆炸集成、一次性组装或整体拼装。这种集成测试策略的做法就是把所有通过单元测试的模块一次性集成到一起进行测试，不考虑组件之间的互相依赖性及可能存在的风险。因此，该方法只适合于规模较小的系统。如图 4.2 所示的程序结构，

先对模块 A、B、C、D、E、F、G 分别做单元测试，再将通过单元测试的各模块一次性集成到一起进行集成测试，采用非渐增式集成的测试方法如图 4.3 所示。

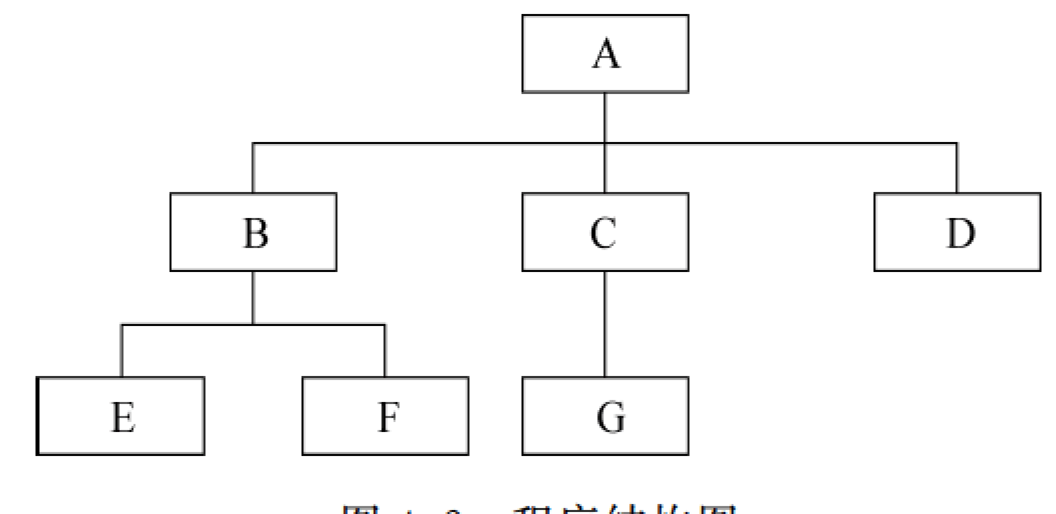


图 4.2 程序结构图

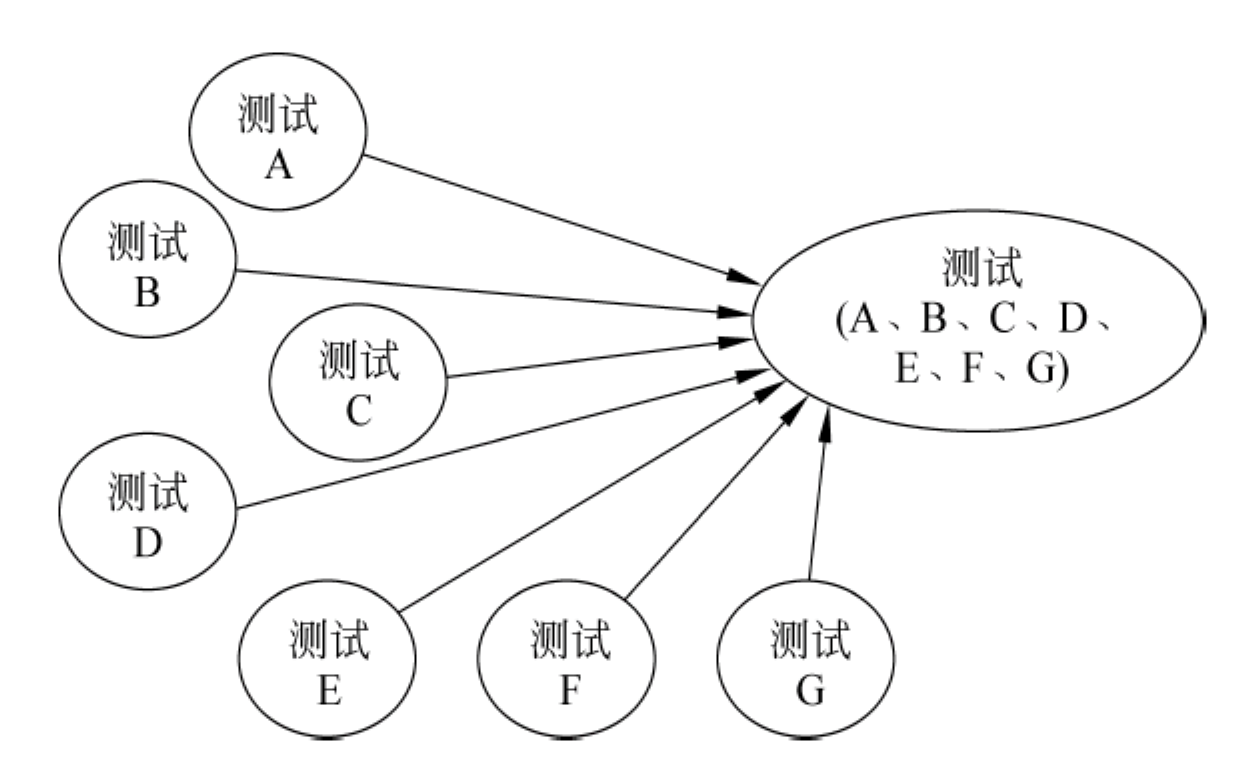


图 4.3 非渐增式集成测试

非渐增式集成的优点如下：

(1) 可以并行调试所有模块。

(2) 需要的测试用例数目少。

(3) 测试方法简单、易行。

非渐增式集成的缺点如下：

(1) 不能对各个模块之间的接口进行充分测试。

(2) 不能很好地对全局数据结构进行测试。

(3) 如果一次集成的模块数量多，集成测试后可能会出现大量的错误。对错误的定位很困难，另外，修改了一处错误之后，很可能新增更多的新错误，新旧错误混杂，给程序的完善带来很大的麻烦。

(4) 即使集成测试通过，也会遗漏很多错误。

非渐增式集成的适用范围如下：

(1) 只需要修改或增加少数几个模块的前期产品稳定的项目。

(2) 功能少，模块数量不多，程序逻辑简单，并且每个组件都已经通过充分的单元测试的小型项目。

(3) 基于严格的净室软件工程(由 IBM 公司开创的开发零缺陷或接近零缺陷的软件的成功做法)开发的产品，并且在每个开发阶段，产品质量和单元测试质量都相当高的产品。

4.2.2 渐增式集成测试

渐增式集成测试与“一步到位”的非渐增式集成测试相反，它把程序划分成小段来构造

和测试，在这个过程中比较容易定位和改正错误；对接口可以进行更彻底的测试；可以使用系统化的测试方法。因此，目前在进行集成测试时普遍采用渐增式集成方法。渐增式集成测试方法不是独立地测试每个单元，而是首先把下一个要被测试的单元同已经测试过的单元集合组装起来，然后再测试，在组装的过程中边连接边测试，以发现连接过程中产生的问题，最后通过渐增式方法逐步组装成要求的软件系统。渐增式测试方法的集成过程有不同的集成策略，典型的有自顶向下和自底向上两种策略。在渐增式集成中要用到两个概念：驱动模块（driver）和桩模块（stub）。其中，驱动模块用以模拟待测模块的上级模块，驱动模块在测试过程中接收数据，把数据传送给被测模块，启动被测模块；桩模块也称存根模块，用以模拟待测模块工作过程中所调用的模块，桩模块由被测模块调用，一般只进行很少的数据处理，如打印、返回等。

1. 自顶向下集成测试

自顶向下集成测试是按照程序和控制结构从主控模块开始，向下逐个把模块连接起来。把附属于主控模块的子模块、孙模块等组装起来的方式有两种：深度优先和广度优先。自顶向下集成测试需要写桩模块。自顶向下渐增式集成测试的步骤如下：

第一步，对主控模块进行测试，测试时用桩模块代替所有直接附属于主控模块的模块。

第二步，根据选定的结合策略（深度优先或广度优先），每次用一个实际模块代换一个桩模块。

第三步，在结合进一个模块的同时进行测试。

第四步，为了保证加入模块没有引进新的错误，可能需要进行回归测试。

第五步，从第二步开始不断重复进行上述过程，直到构造起完整的软件结构为止。

图4.2的程序结构采用自顶向下、深度优先的集成测试过程，如图4.4所示。

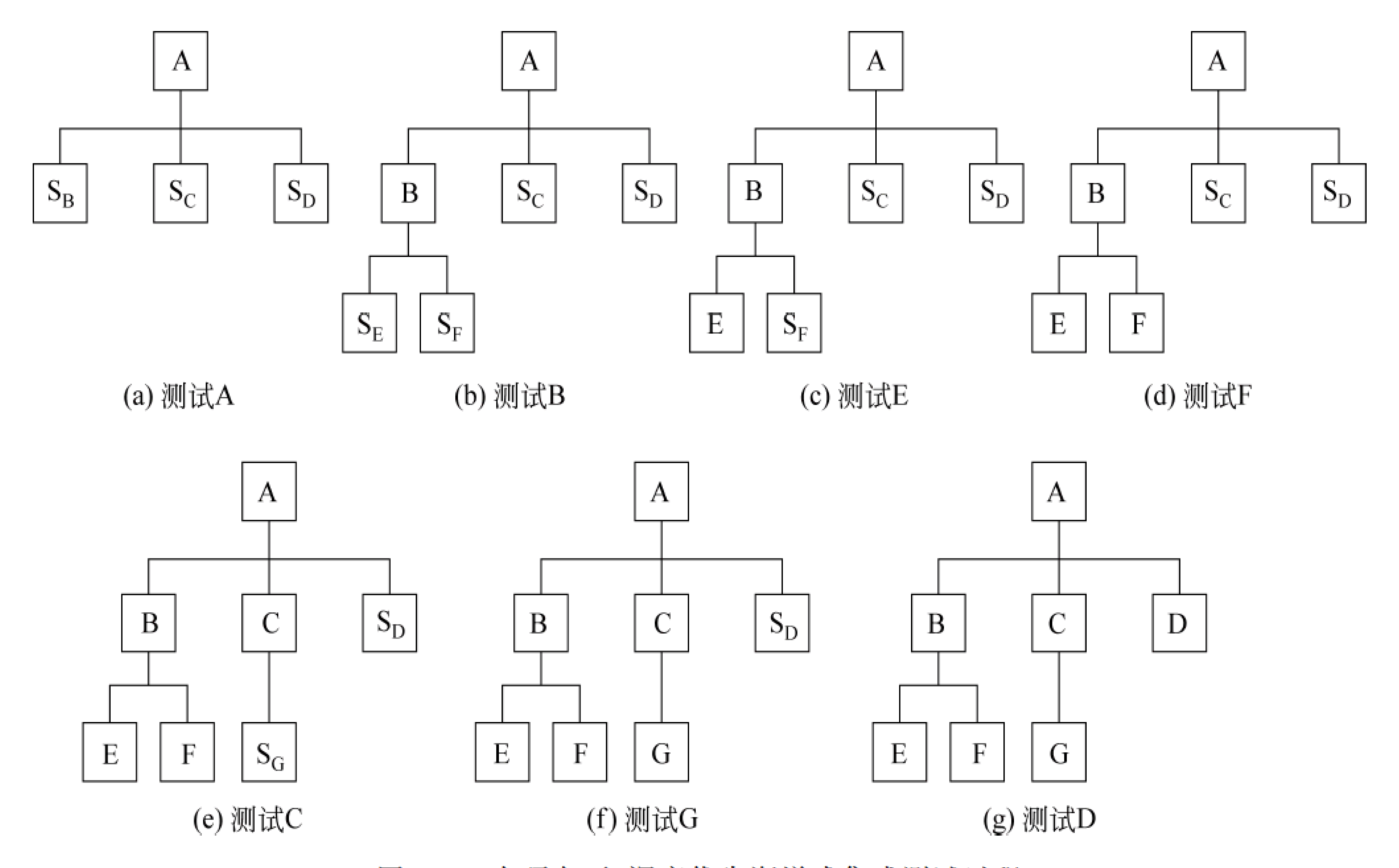


图4.4 自顶向下、深度优先渐增式集成测试过程

如果采用广度优先的集成策略，各模块集成的顺序为 A→B→C→D→E→F→G。

自顶向下集成测试的优点如下：

(1) 在测试的过程中,可以较早地验证主要的控制和判断点。

(2) 选择深度优先组合方式,可以首先实现和验证一个完整的软件功能,可先对逻辑输入的分支进行组装和测试,检查和克服潜藏的错误和缺陷。

(3) 验证其功能的正确性,为此后主要分支的组装和测试提供保证。

(4) 能够较早地验证功能可行性,给开发者和用户带来成功的信心。

(5) 只有在个别情况下,才需要驱动程序(最多不超过一个),减少了测试驱动程序开发和维护的费用。

(6) 可以和开发设计工作一起并行执行集成测试,能够灵活地适应目标环境。

(7) 容易进行故障隔离和错误定位。

自顶向下集成的缺点如下：

(1) 在测试时需要为每个模块的下层模块提供桩模块,桩模块的开发和维护费用大。

(2) 底层组件的需求变更可能会影响到全局组件,需要修改整个系统的多个上层模块。

(3) 要求控制模块具有比较高的可测试性。

(4) 可能会导致底层模块特别是被重用的模块测试不够充分。

自顶向下集成的适用范围如下：

(1) 控制结构比较清晰和稳定的应用程序。

(2) 系统高层的模块接口变化的可能性比较小。

(3) 产品的低层模块接口还未定义或可能会经常因需求变更等原因被修改。

(4) 产品中的控制模块技术风险较大,需要尽可能提前验证。

(5) 需要尽早看到产品的系统功能行为。

(6) 在极限编程(extreme programming)中使用测试优先的开发方法。

2. 自底向上集成测试

自底向上集成方式从程序模块结构中最底层的模块开始组装和测试。因为模块是自底向上进行组装的,对于一个给定层次的模块,它的子模块(包括子模块的所有下属模块)事前已经完成组装并经过测试,所以不再需要编制桩模块。自底向上集成测试需要编写驱动模块。自底向上渐增式集成测试的步骤如下：

第一步,把低层模块组合成实现某个特定软件子功能的簇。

第二步,编写一个驱动程序,协调测试数据的输入和输出。

第三步,对由模块组成的子功能簇进行测试。

第四步,去掉驱动程序,沿软件结构自下向上移动,把子功能簇组合起来形成更大的子功能簇。

图 4.2 的程序结构采用自底向上的集成测试过程如图 4.5 所示。

自底向上集成测试的优点如下：

(1) 即使数据流并未构成有向的非环状图,生成测试数据也没有困难。

(2) 可以尽早验证底层模块的行为。

(3) 对实际被测模块的可测试性要求较少。

(4) 减少了桩模块的工作量。

(5) 容易对错误进行定位。

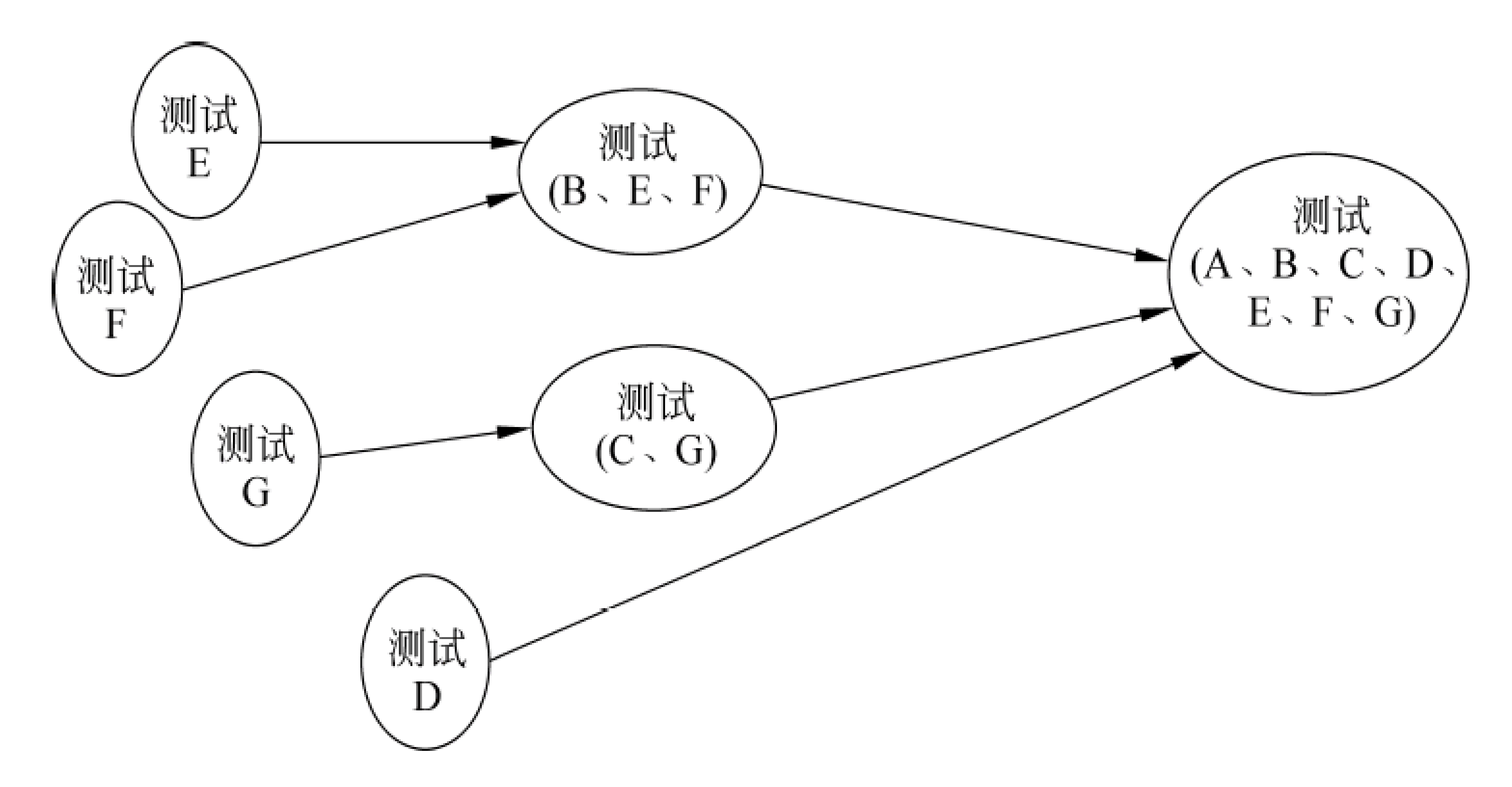


图 4.5 自底向上集成测试过程

自底向上集成测试的缺点如下：

(1) 直到最后一个模块加进去之后才能看到整个系统的框架。

(2) 只有到测试过程的后期才能发现时序问题和资源竞争问题。

(3) 驱动模块的设计工作量大。

(4) 不能及时发现高层模块设计上的错误。

自底向上集成测试的适用范围如下：

(1) 底层模块接口比较稳定的产品。

(2) 高层模块接口变更比较频繁的产品。

(3) 底层模块开发和单元测试工作完成较早的产品。

4.2.3 三明治集成测试

三明治集成测试是结合了自顶向下和自底向上两种测试策略，在顶层使用自顶向下策略，在底层使用自底向上策略。这种策略的关键就是如何选取程序结构图中的基准层(或标准层)，在基准层下方使用自底向上的策略，上方使用自顶向下的策略。基准层的选择对于整个测试工作有较大的影响。三明治集成测试的步骤如下：

第一步，确定以哪一层为基准层来进行集成。

第二步，对基准层下面的各层使用自底向上的集成策略。

第三步，对基准层上面的各层使用自顶向下的集成策略。

第四步，对基准层各模块与相应的下层集成。

第五步，对系统进行整体测试。

在如图 4.6 所示的程序结构图中，选择不同的基准层，测试的过程是不同的。

选择 BCDE 层为基准层，采用三明治集成策略进行集成测试的过程如图 4.7 所示。

选择 FGHIJK 层为基准层，采用三明治集成策略进行集成测试的过程如图 4.8 所示。

三明治集成测试的优点如下：除了具有自顶向下和自底向上两种集成策略的优点之外，运用

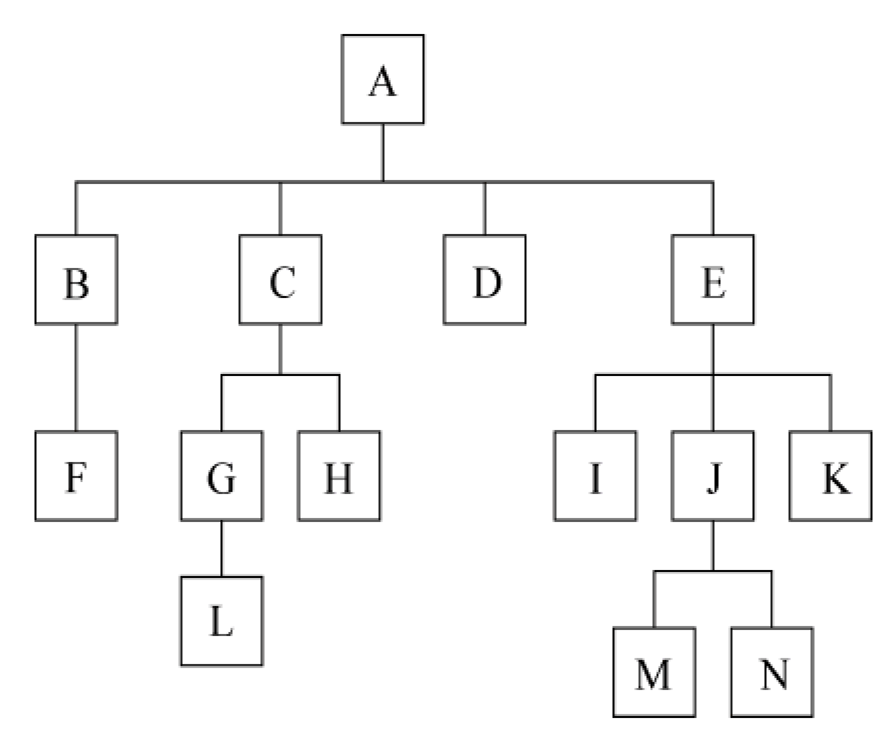


图 4.6 复杂的程序结构图

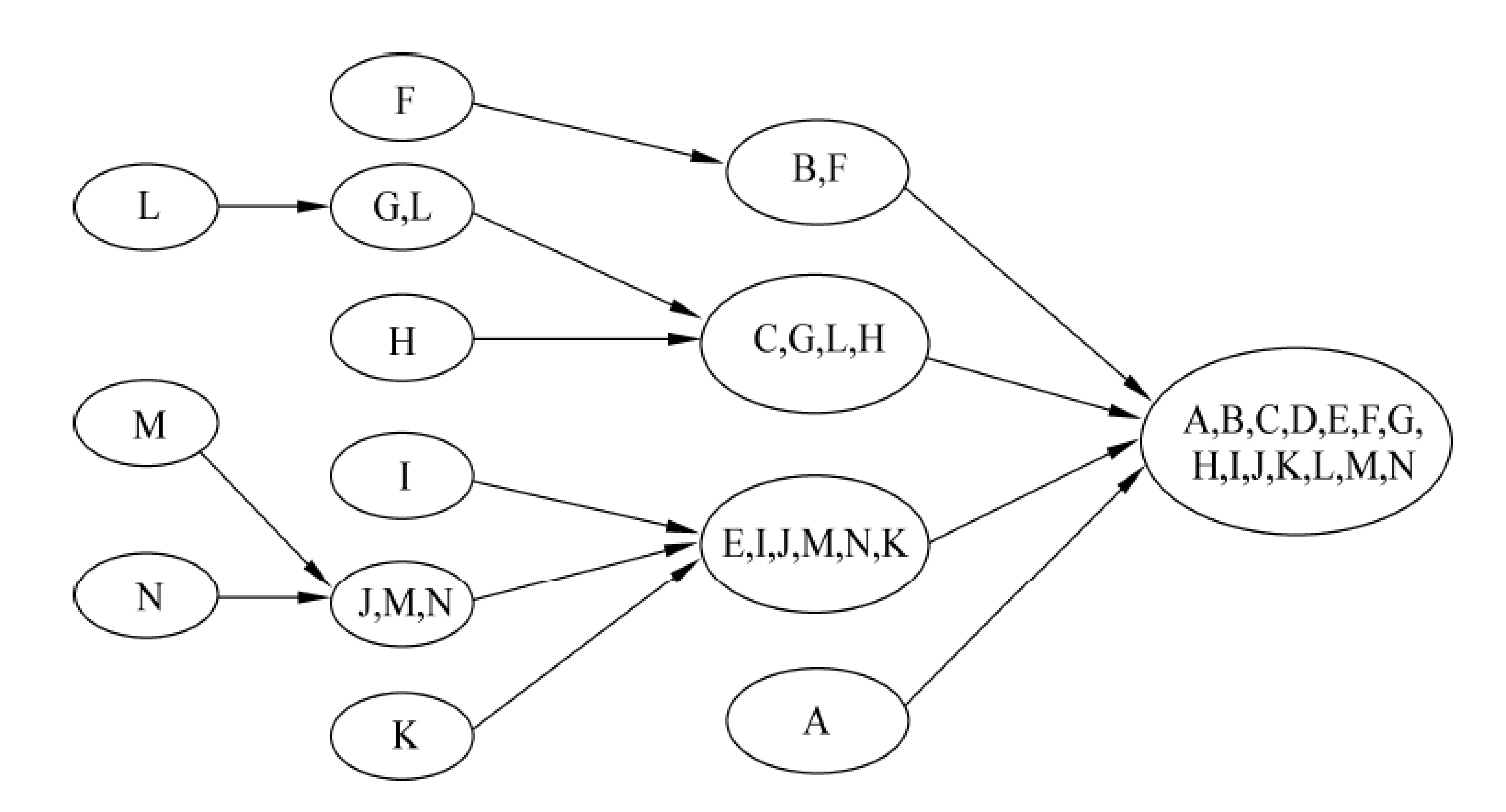


图 4.7　选择 BCDE 层为基准层的集成过程

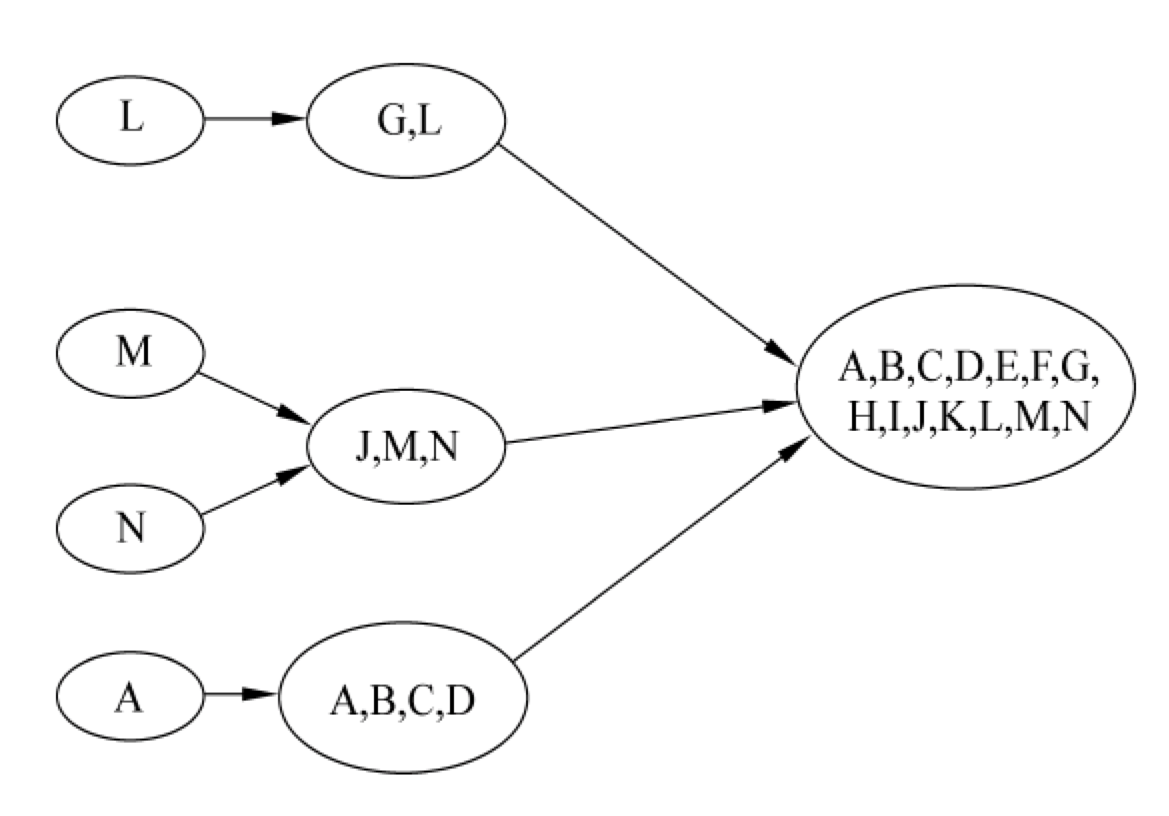


图 4.8　选择 FGHIJK 层为基准层的集成过程

一定的技巧,能够减少桩模块和驱动模块的开发工作量。

其缺点是:在被集成之前,中间层不能尽早得到充分的测试。

其适用范围:多数软件开发项目都可以应用此集成测试策略。

4.3　集成测试过程

按照集成测试不同阶段的任务,可以将集成测试的整个过程划分为5个阶段,依次是制定集成测试计划、设计集成测试、实施集成测试、执行集成测试和评估集成测试。集成测试的过程如图4.9所示。

4.3.1　制定集成测试计划

集成测试计划阶段开始于概要设计评审通过后约一个星期,可参考需求规格说明书、概要设计文档和产品开发计划表制定。集成测试计划阶段的工作如下:

(1) 确定被测试对象和测试范围。

(2) 评估集成测试被测试对象的数量及难度,即工作量。

(3) 确定角色分工和划分工作任务。

(4) 标示出测试各个阶段的时间、任务和约束条件。

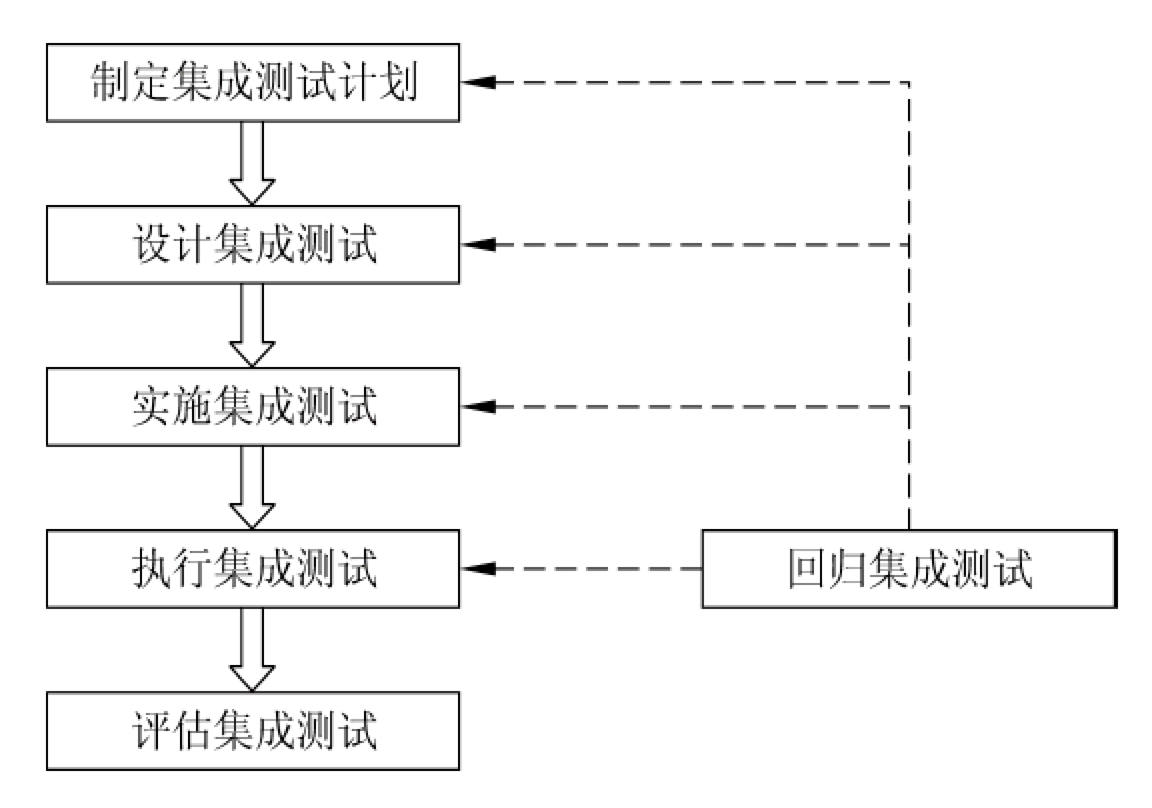


图 4.9 集成测试过程

(5) 考虑一定的风险分析及应急计划。

(6) 考虑和准备集成测试需要的测试工具、测试仪器和环境等资源。

(7) 考虑外部技术支持的力度、深度以及相关培训安排,定义测试完成的标准。

4.3.2 设计集成测试

集成测试设计阶段一般在详细设计开始时,把需求规格说明书、概要设计和集成测试计划文档作为重要的依据。集成测试设计阶段的工作如下:

(1) 被测对象结构分析。

(2) 集成测试模块分析。

(3) 集成测试接口分析。

(4) 集成测试策略分析。

(5) 集成测试工具分析。

(6) 集成测试环境分析。

(7) 集成测试工作量估计和安排。

4.3.3 实施集成测试

集成测试实施阶段主要在详细设计阶段的评审已经通过后,参考需求规格说明书、概要设计、集成测试计划和集成测试设计文档进行。该阶段的工作如下:

(1) 集成测试用例设计。

(2) 集成测试规程设计。

(3) 集成测试代码设计。

(4) 集成测试脚本开发。

(5) 集成测试工具开发或选择。

4.3.4 执行集成测试

测试人员在单元测试完成以后就可以执行集成测试。按照相应的测试规程,借助集成测试工具,并把需求规格说明书、概要设计、集成测试计划、集成测试设计、集成测试用例、集成测试规程、集成测试代码和集成测试脚本作为测试执行的依据,执行集成测试用例。测试

执行的前提条件就是单元测试已经通过评审。当测试执行结束后，测试人员要记录每个测试用例执行后的结果，填写集成测试报告，最后提交给相关人员进行评审。

4.3.5 评估集成测试

当集成测试执行结束后，要召集相关人员，如测试设计人员、编码人员和系统设计人员等，对测试结果进行评估，确定是否通过集成测试。例如，是否成功地执行了测试计划中规定的所有集成测试，是否修正了所发现的错误，测试结果是否通过了专门小组的评审。

集成测试应由专门的测试小组来进行，测试小组由有经验的系统设计人员和程序员组成。整个测试活动要在评审人员出席的情况下进行。

在完成预定的测试工作之后，测试小组应负责对测试结果进行整理和分析，形成测试报告。测试报告中要记录实际的测试结果、在测试中发现的问题、解决这些问题的方法以及问题解决之后再次测试的结果。此外还应提出目前不能解决、还需要管理人员和开发人员注意的一些问题，供测试评审和最终决策提出处理意见。

4.4 集成测试用例设计方法

以上所介绍的几种集成测试方法都是“基于功能分解”的集成，也就是它们都需要依赖于程序结构图，这也是基于分解的集成测试方法的缺点，采用基于调用图的集成测试方法和基于路径的集成测试方法在一定程度上可以避免对程序结构图的依赖。

4.4.1 基于调用图的集成测试

基于调用图的集成测试方法将测试的角度从程序结构图转换到程序中模块之间的调用关系上。基于调用图的集成测试方法主要包括成对集成和相邻集成。调用图的形式如图4.10所示。

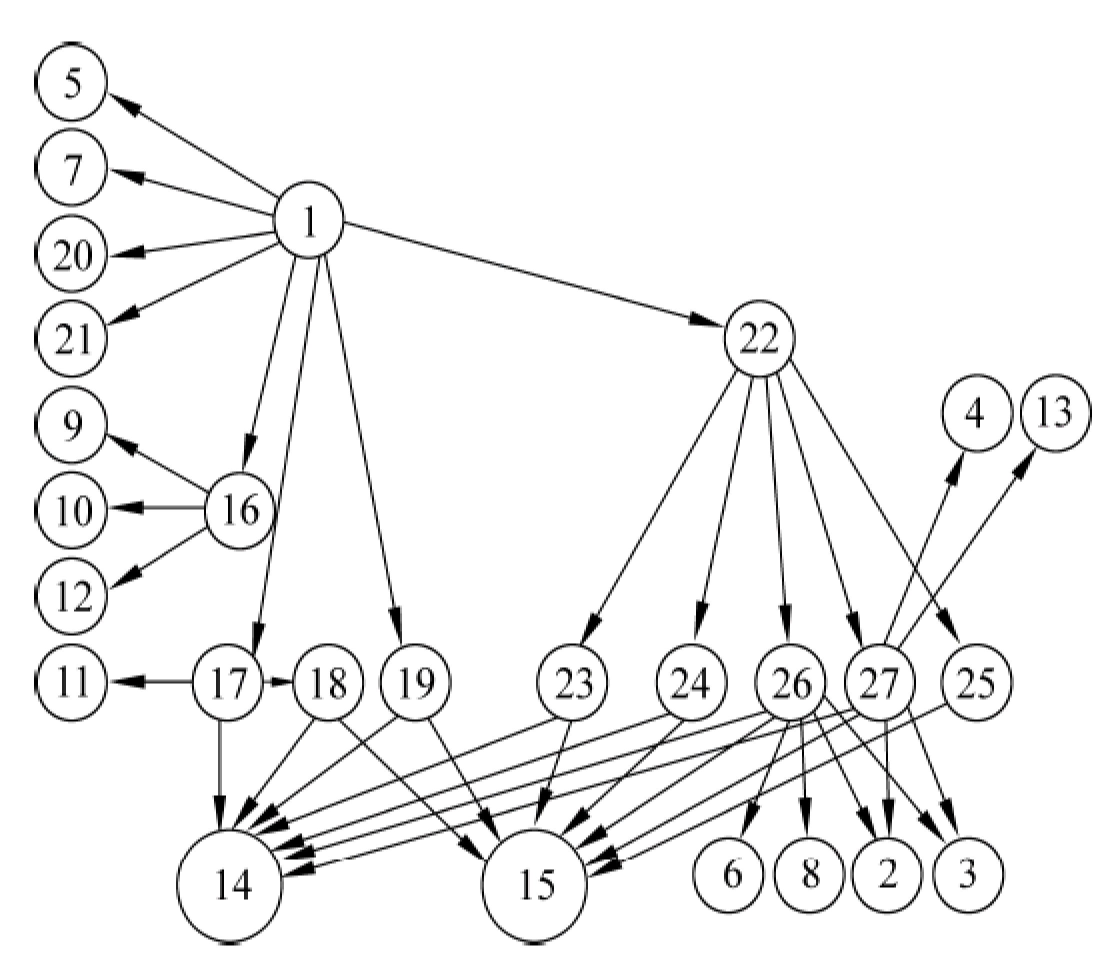


图4.10 调用图

1. 成对集成

成对集成思想的提出，主要是为了减少桩模块和驱动模块的开发工作。在成对集成中，

可采用调用对的测试方法，即成对集成是将集成限定在调用图中的一对测试单元中，最终可使调用图中的每一条边形成一个集成测试会话。虽然工作量比较大，但是极大地减少了桩模块和驱动模块的开发工作量。图 4.10 的调用图共有 40 个集成测试会话(40 条边)，图 4.11 标出了部分集成测试会话。

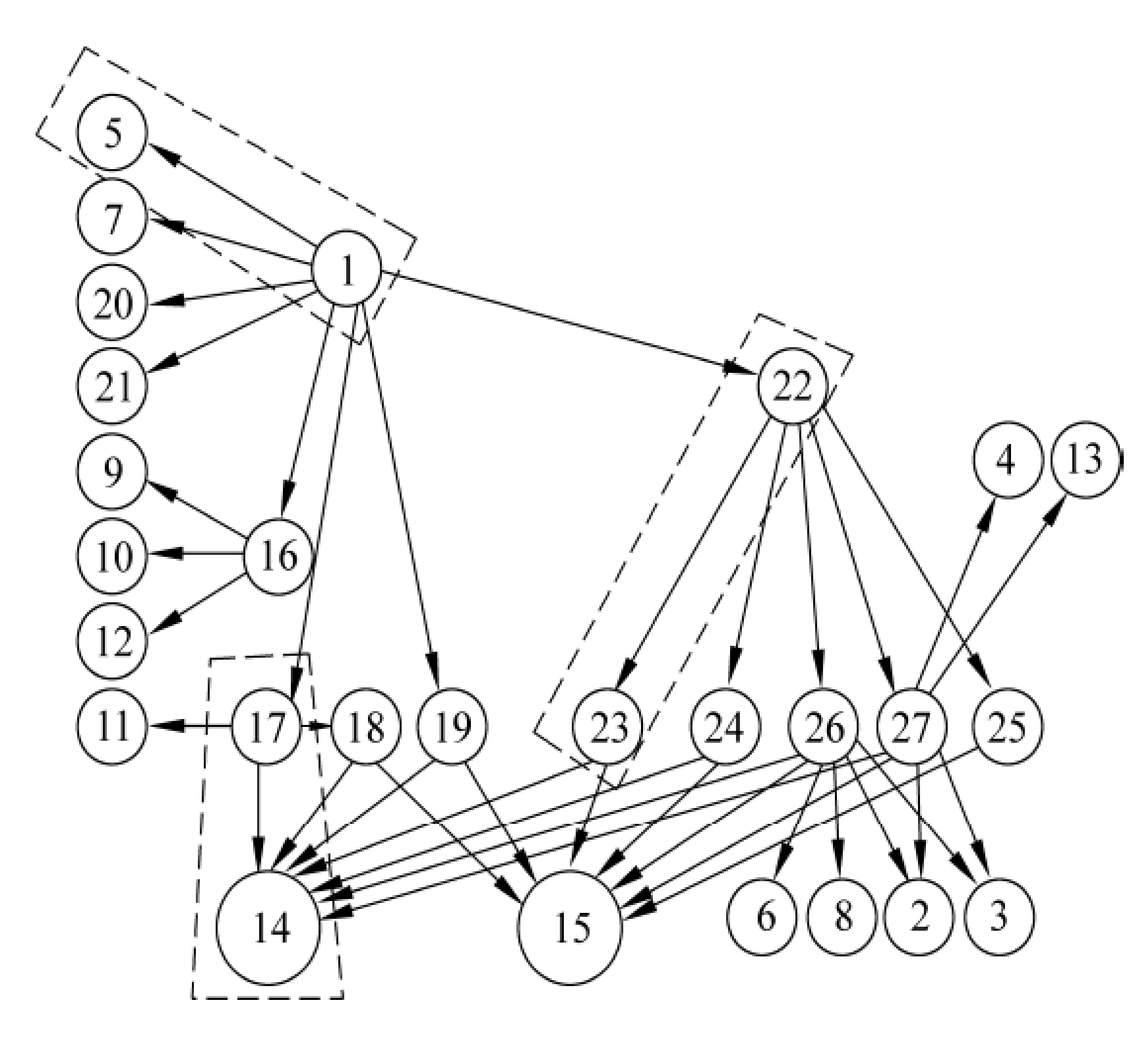


图 4.11　成对集成

2. 相邻集成

在成对集成中，集成测试会话比较多。为了减少测试的数量，在调用图集成测试中相邻集成是一种不错的测试方法。相邻集成主要是以相邻节点为集合进行测试，相邻节点是指有向图中所有的直接前趋节点和后继节点。图 4.10 的相邻集成测试的邻居集合共有 11 个，即，去掉所有叶子节点的相邻节点集合，如图 4.12 所示。表 4.1 列出了图 4.12 中所有标出的节点的邻居节点所形成的集合。

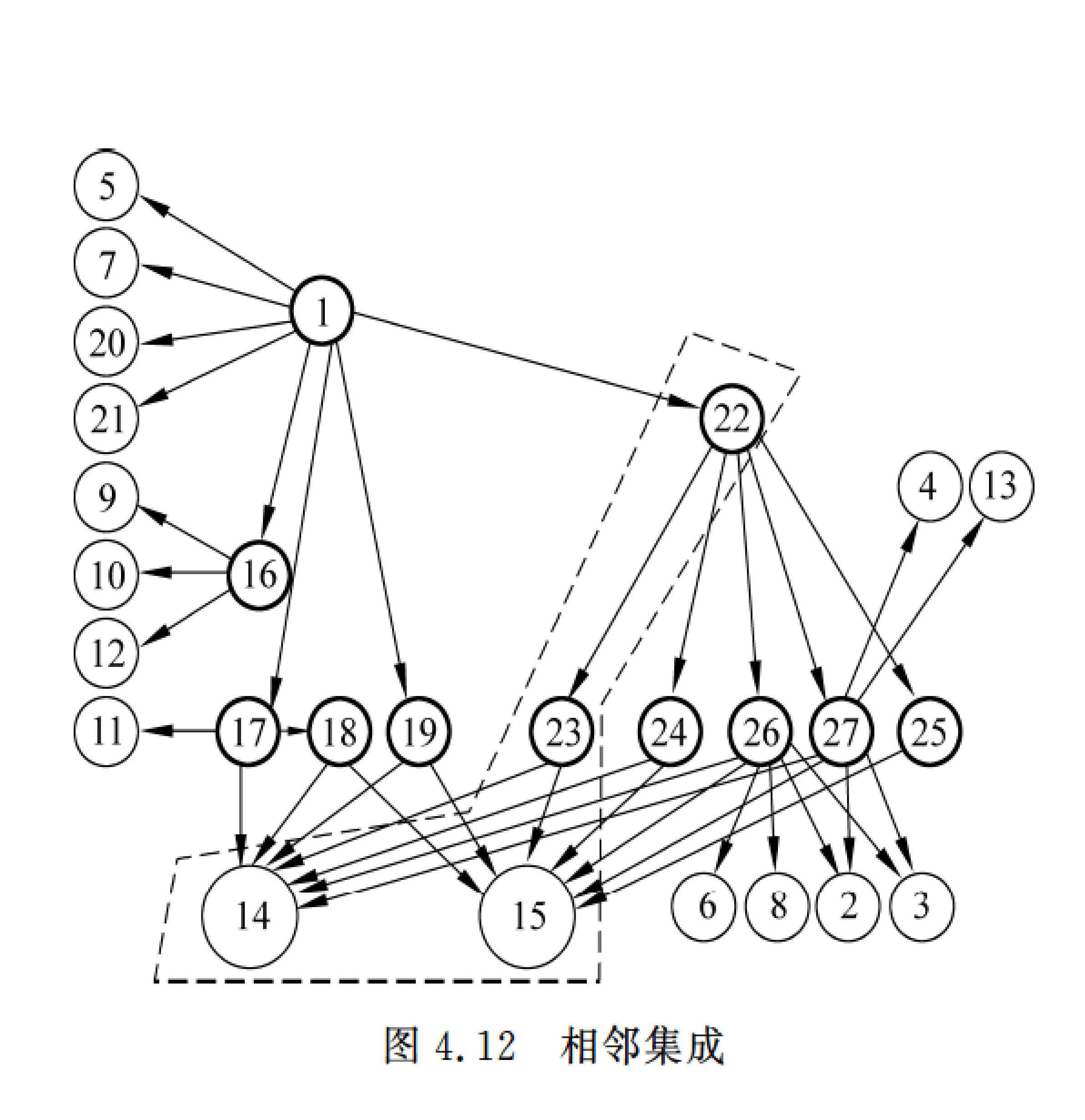


图 4.12　相邻集成

表 4.1　图 4.12 的相邻节点集合

节点	前趋节点	后继节点
1	无	5,7,20,21,16,17,19,22
16	1	9,10,12
17	1	11,14,18
18	17	14,15
19	1	14,15
22	1	23,24,26,27,25
23	22	14,15
24	22	14,15
25	22	15
26	22	14,15,6,8,2,3
27	22	14,15,2,3,4,13

基于调用图的集成测试方法避免了对程序结构的依赖,而以程序调用为基础,减少了桩模块和驱动模块的开发工作量;基于调用图的集成与以构建和合成为特征的开发匹配得很好。这种方法最大的缺点就是缺陷隔离存在问题,缺陷难以定位。

4.4.2 基于 MM 路径的集成测试

在单元测试中,采用路径覆盖的测试方法可以遍历源程序中所有可能的路径。在集成测试中也有类似的测试方法,即 MM 路径测试方法。这种方法是由 Paul C. Jorgensen 提出的,MM 是 Message-Method 的简称。MM 路径可以描述单元之间控制转移的执行路径序列。下面先介绍 MM 路径测试方法中相关的概念。

源节点:程序开始或重新开始处的语句片段。

汇节点:程序执行结束处的语句片段。

模块执行路径:以源节点开始,以汇节点结束的语句序列,其间没有插入汇节点。

消息:一种程序设计机制,通过该机制可将控制从一个单元转移到另一个单元。

MM 路径:模块执行路径和消息穿插出现的序列。

MM 路径示例如图 4.13 中的粗实线所示。该图中有 3 个模块,模块 A 调用模块 B,模块 B 又调用模块 C。其中,模块 A 的源节点是 1 和 5,汇节点是 4 和 6;模块 B 的源节点是 1 和 3,汇节点是 2 和 4;模块 C 的源节点是 1,汇节点是 5。

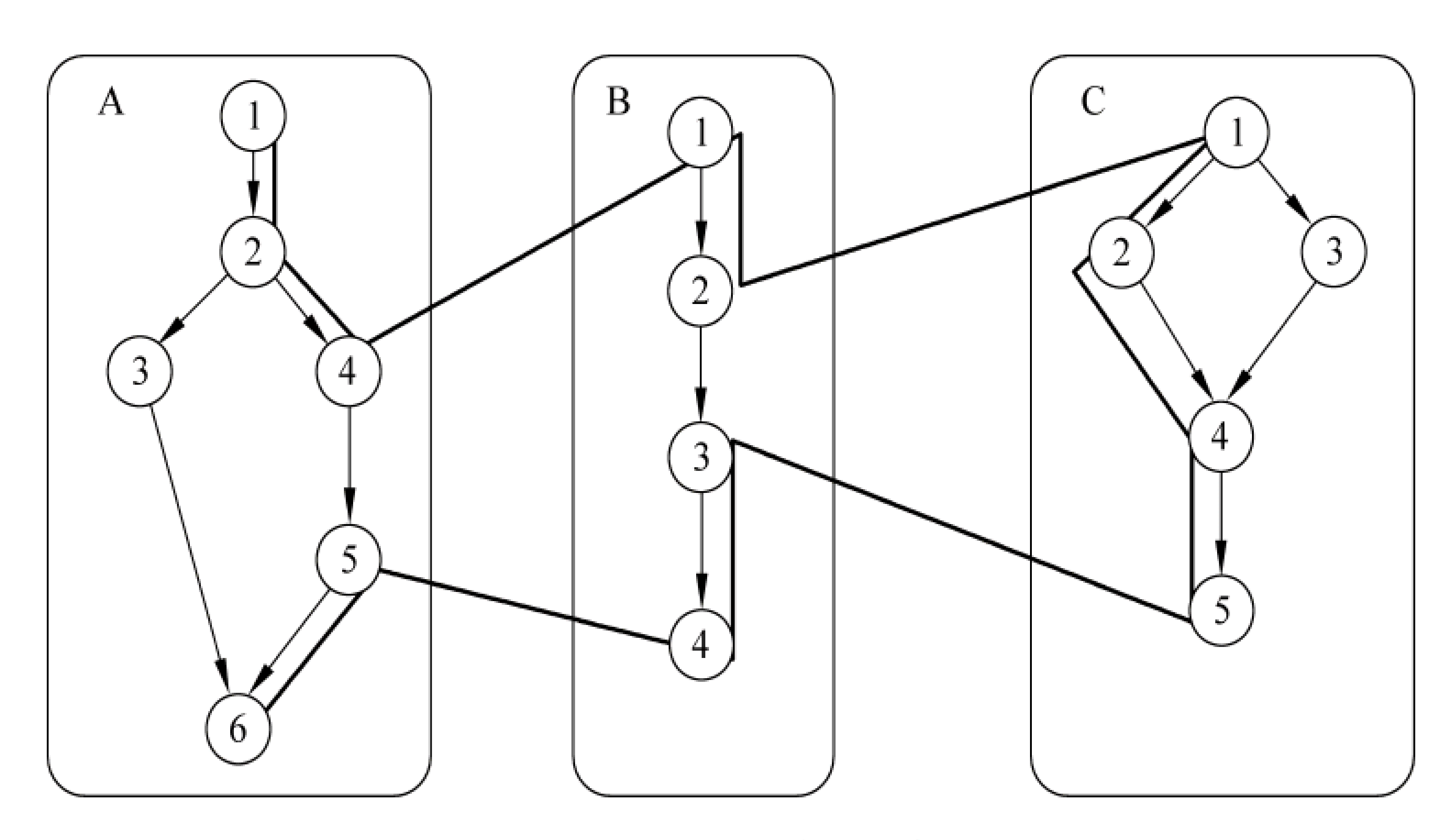


图 4.13 MM 路径示例

模块 A 的执行路径:MEP(A,1)=<1,2,3,6>,MEP(A,2)=<1,2,4>,MEP(A,3)=<5,6>;

模块 B 的执行路径:MEP(B,1)=<1,2>,MEP(B,2)=<3,4>;

模块 C 的执行路径:MEP(C,1)=<1,2,4,5>,MEP(C,2)=<1,3,4,5>。

图 4.13 的 MM 路径如图 4.14 所示。

MM 路径集成测试是功能测试和结构测试的结合,从输入和输出来看,它具有功能测试的特点;从表示方式来看,它具有结构测试的特点。基于 MM 路径的集成的优势在于它与实际系统的行为紧密相连,而不是依赖于基于功能分解和基于调用图的结构性测试。但是这种测试的主要缺点是工作量比较大。MM 路径测试的最低要求为系统中的所有消息至少均被覆盖一次。

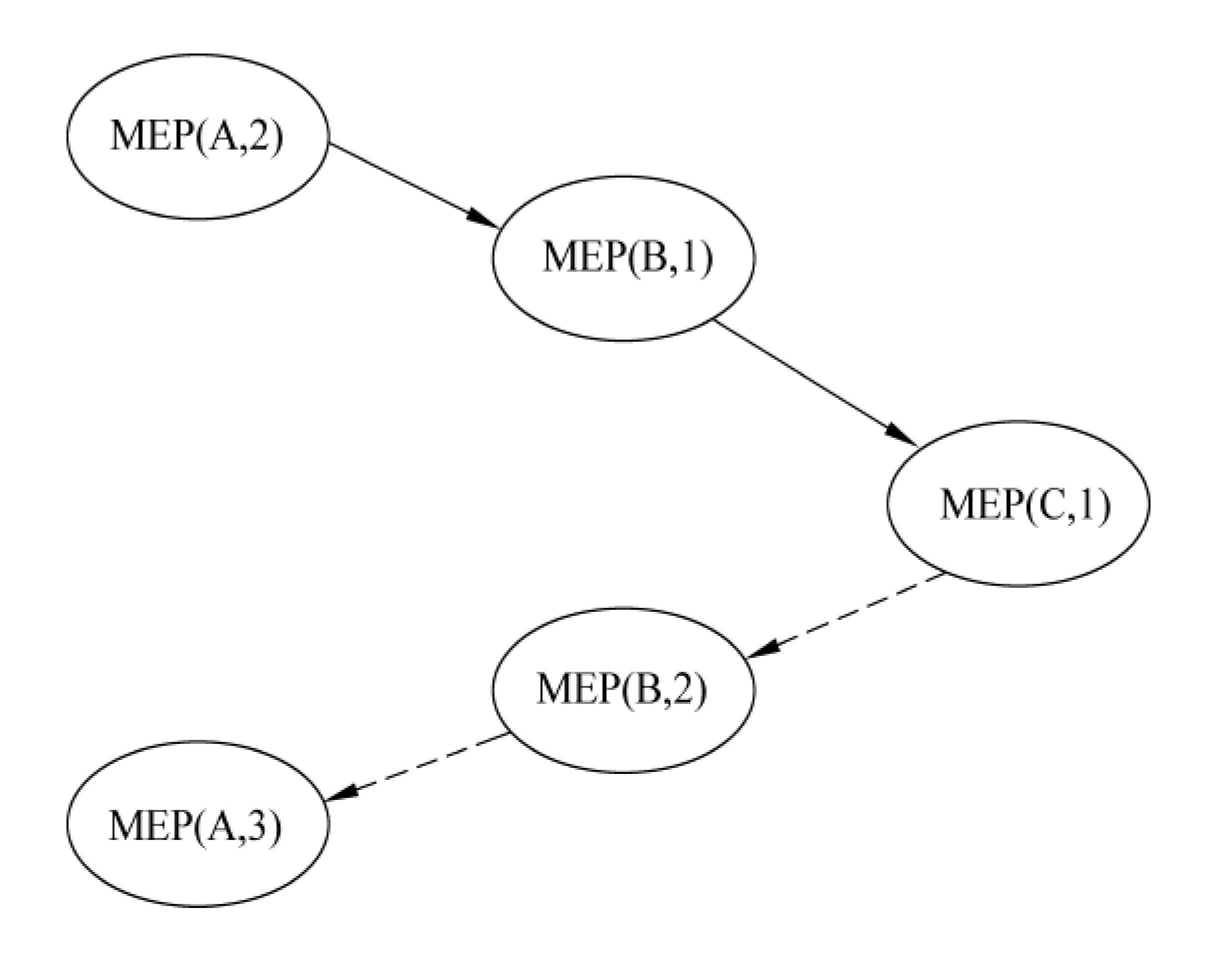


图 4.14　MM 路径

4.4.3　案例分析——NextDate 集成测试用例设计

1. 问题分析及源码示例

NextDate(month,day,year)实现输出给定日期的下一个日期,其中 1≤month≤12,1≤day≤31,1900≤year≤2050,现在重写 2.3 节的 NextDate 程序。

NextDate 由 5 个类实现,5 个类之间通过彼此发送消息进行交互,其中:

CalendarUnit 是一个抽象类;

Date 是 CalendarUnit 的一个子类;

Day 是 CalendarUnit 的一个子类;

Month 是 CalendarUnit 的一个子类;

Year 是 CalendarUnit 的一个子类。

抽象类 CalendarUnit.java 的源码如下:

```
public abstract class CalendarUnit {
    int currentPos;
    public void setCurrentPos(int pCurrentPos){
        this.currentPos=pCurrentPos;
    }
    abstract protected boolean increment();
}
```

Date.java 的源码如下:

```
public class theDate extends CalendarUnit{
    private Day d;
    private Month m;
    private Year y;
    public Date(int pDay,int pMonth,int pYear){
        this.d=new Day(pDay,m);
        this.m=new Month(pMonth,y);
        this.y=new Year(pYear);
    }
```

```
        public boolean increment(){
        if(!(d.increment()))
            if(!(m.increment())){
                y.increment();
                m.setMonth(1,y);
            }else
            d.setDay(1,m);
        return true;
    }
    public String printDate(){
        return new Integer(m.getMonth()).toString()+"/"
                +new Integer(d.getDay()).toString()+"/"
                +new Integer(y.getYear()).toString();
    }
}
```

Day.java 的源码如下：

```
public class Day extends CalendarUnit{
    private Month m;
    //private int currentPos;
    public Day(int pDay,Month pMonth){
        setDay(pDay,pMonth);
    }
    public void setDay(int pDay, Month pMonth) {
        setCurrentPos(pDay);
        this.m=pMonth;
    }
    public int getDay(){
        return currentPos;
    }
    public boolean increment(){
        currentPos=currentPos+1;
        if(currentPos<=m.getMonthSize())
            return true;
        else{
            currentPos=1;
            return false;
        }
    }
}
```

Month.java 的源码如下：

```
public class Month extends CalendarUnit{
    private Year y;
    private int sizeIndex[]={31,28,31,30,31,30,31,31,30,31,30,31};
    private int currentPos;
```

```
    public Month(int pcur,Year pYear){
        setMonth(pcur,pYear);
    }
    public void setMonth(int pcur, Year pYear) {
        setCurrentPos(pcur);
        this.y=pYear;
    }
    public int getMonth(){
        return currentPos;
    }
    public int getMonthSize(){
        if(y.isLeap()){
            sizeIndex[1]=29;
        }
        else
            sizeIndex[1]=28;
        return sizeIndex[currentPos-1];
    }
    @Override
    public boolean increment() {
        currentPos=currentPos+1;
        if(currentPos>12)
            return false;
        else
            return true;
    }
}
```

Year.java 的源码如下：

```
public class Year extends CalendarUnit{
    public Year(int pYear){
        setCurrentPos(pYear);
    }
    public int getYear(){
        return currentPos;
    }
    public boolean increment(){
        currentPos=currentPos+1;
        return true;
    }
    public boolean isLeap() {
        if(((currentPos%4==0)&&!(currentPos%100==0))||(currentPos%400==0))
            return true;
        else
            return false;
    }
}
```

在 NextDate 问题中，主要的测试焦点是源码中的 increment 方法，因此，需要设计针对该方法的测试类，该问题中的测试类 TestClass.java 的源码如下：

```
public class TestClass {
    public static void main(String args[]){
        theDate testdate=new theDate(31,1,2013);
        testdate.increment();
        String strNextDate=testdate.printDate();
        System.out.println(strNextDate);
    }
}
```

2. 测试关键问题

NextDate 问题集成测试的关键点在于 theDate 类的 increment()方法，为了覆盖所有的消息，设计了 3 个方法：

testDayIncrement()方法执行 msg7 的 true 分支；

testMonthIncrement()方法执行 msg7 的 false 分支，msg8 和 msg11 的 true 分支；

testYearIncrement()方法执行 msg7 的 false 分支，msg8、msg9 和 msg10 的 false 分支。

通过执行这 3 个方法，各个类间的所有消息发送都能被覆盖到。

NextDate 问题的类结构、类之间的协作关系以及类之间的消息传递如图 4.15 所示。

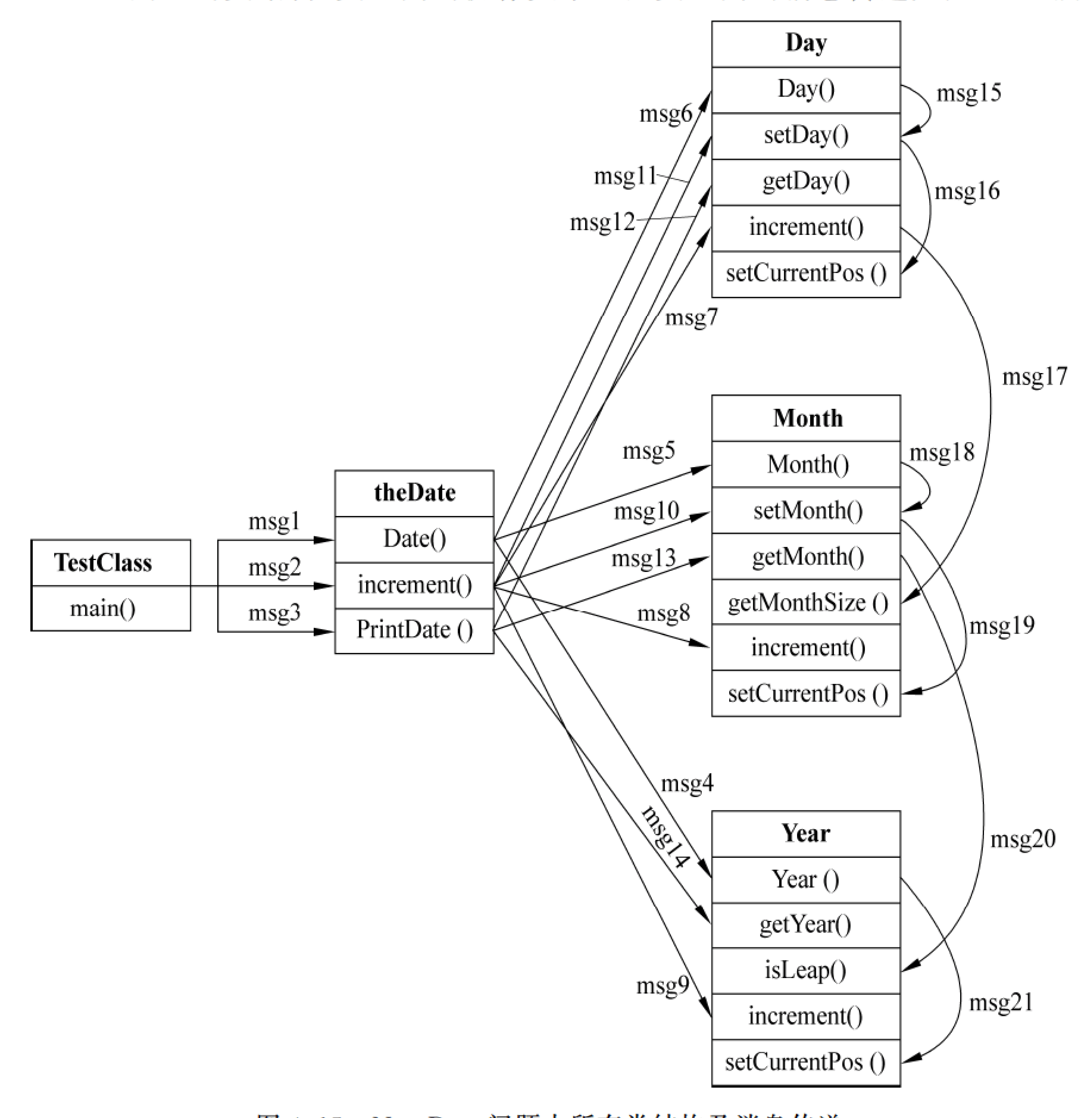


图 4.15 NextDate 问题中所有类结构及消息传递

3. 各方法的 MM 路径及覆盖消息说明

testDayIncrement()方法、testMonthIncrement()方法和 testYearIncrement()方法的 MM 路径测试覆盖消息情况如图 4.16、图 4.17 和图 4.18 所示。其中，图 4.16 是用于测试 Day 的 increment()方法的 MM 路径图，选择的测试用例为 day＝13，month＝1，year＝2013，其中没有覆盖 msg8、msg9、msg10 和 msg11；图 4.17 是用于测试 Month 的 increment()方法的 MM 路径图，选择的测试用例为 day＝31，month＝1，year＝2013，其中没有覆盖 msg9 和 msg10；图 4.18 是用于测试 Year 的 increment()方法的 MM 路径图，选择的测试用例为 day＝31，month＝12，year＝2013，其中覆盖了剩余没有被覆盖的消息。

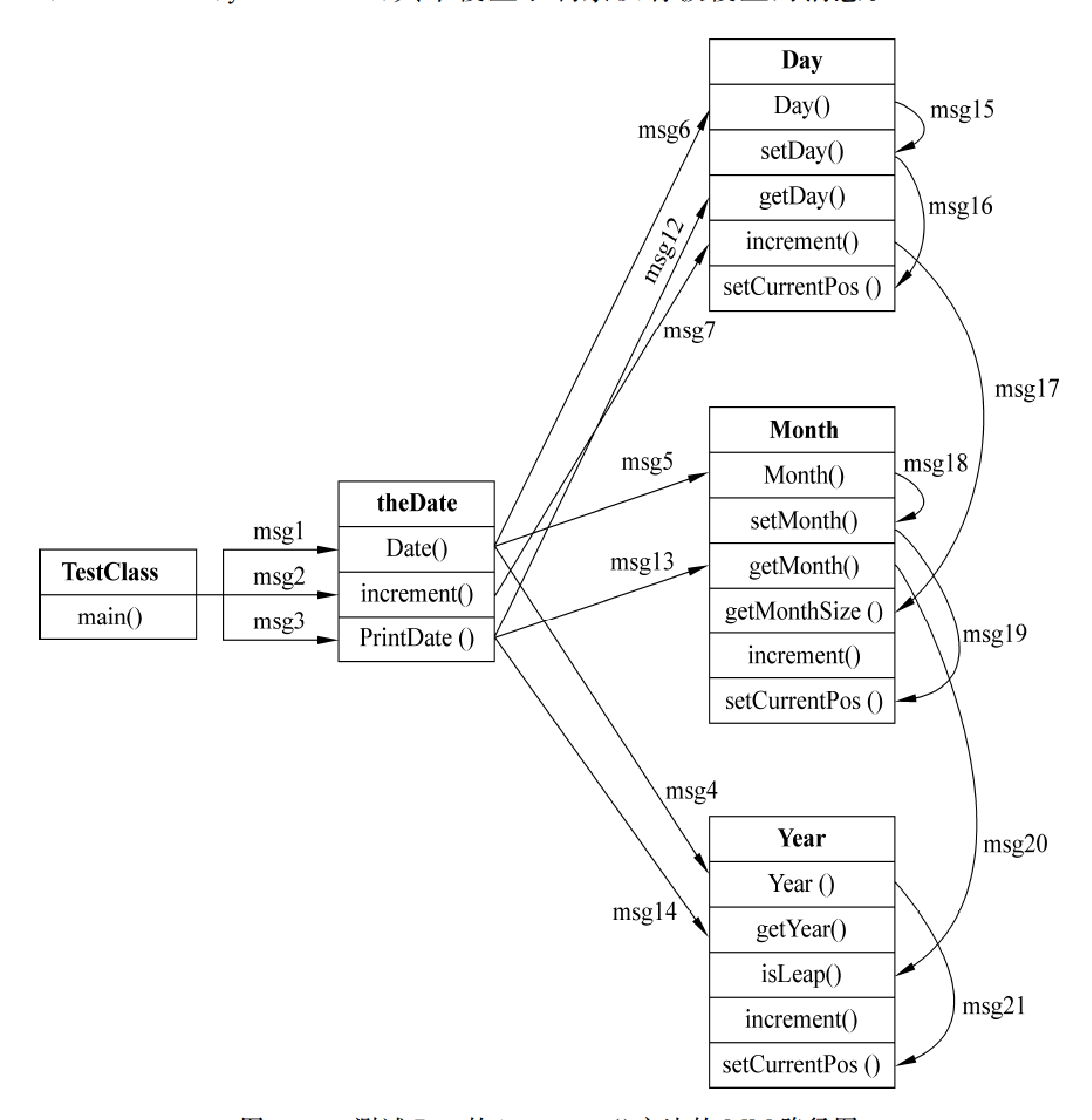


图 4.16 测试 Day 的 increment()方法的 MM 路径图

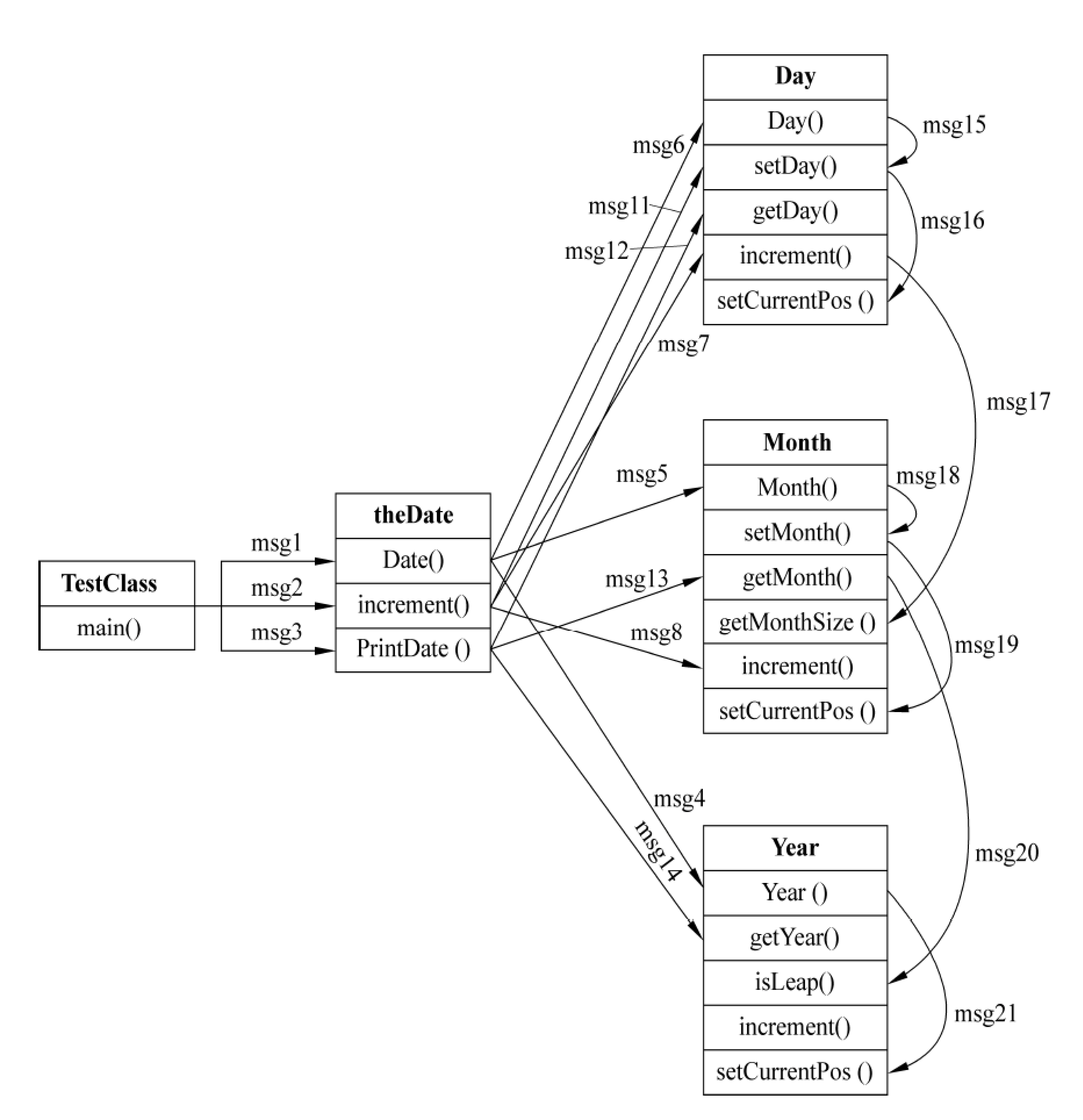


图 4.17 测试 Month 的 increment()方法的 MM 路径图

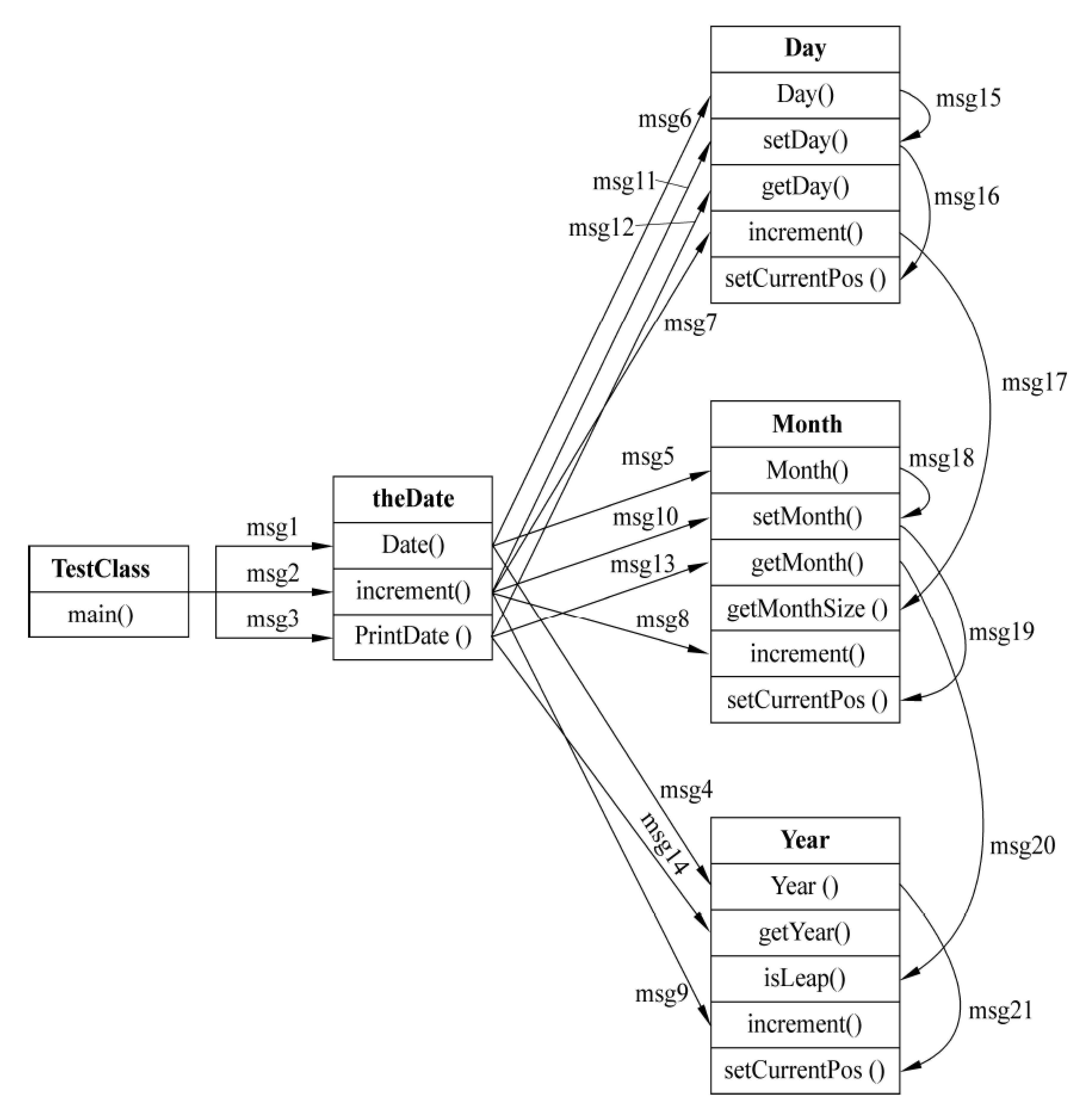


图 4.18 测试 Year 的 increment()方法的 MM 路径图

4.5 本章小结

集成测试和单元测试相比,更容易出现问题,因此,在软件测试过程中,集成测试要比单元测试更复杂一些。本章主要介绍了集成测试的方法,首先介绍了基于功能分解的集成测试,包括渐增式集成测试、非渐增式集成测试和三明治集成测试方法,这些方法都要依赖于程序结构。而后介绍的基于调用图的集成测试方法避免了对程序结构的依赖,转向以程序调用为基础的行为观点,减少了桩模块和驱动模块的开发工作量。最后介绍的 MM 路径集成测试是功能测试和结构测试的结合,其优势在于它与实际系统的行为紧密相连,而不是依赖于基于分解和基于调用图的结构性测试。本章最后以重写的 NextDate 问题为例,介绍了 MM 路径集成测试的实际应用过程。

习　题

1. 判断对错。

(1) 自底向上集成需要测试员编写驱动模块。

(2) 自顶向下集成需要测试员编写桩模块。

(3) 非渐增式集成测试方法发现错误难以诊断定位。

(4) MM 路径是可执行路径。

2. 简述集成测试的过程。

3. 根据如图 4.19 所示的程序模块结构图,画出自顶向下集成测试、自底向上集成测试及三明治集成测试的过程。

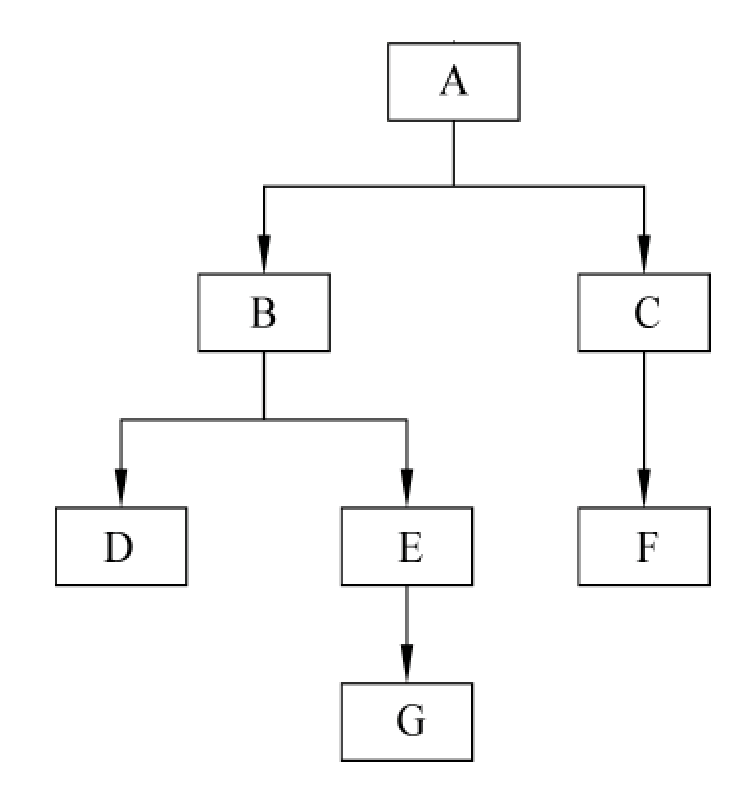


图 4.19 程序模块结构图

4. 根据图 4.20,采用基于调用图的集成测试方法分别进行相邻集成测试和成对集成测试。

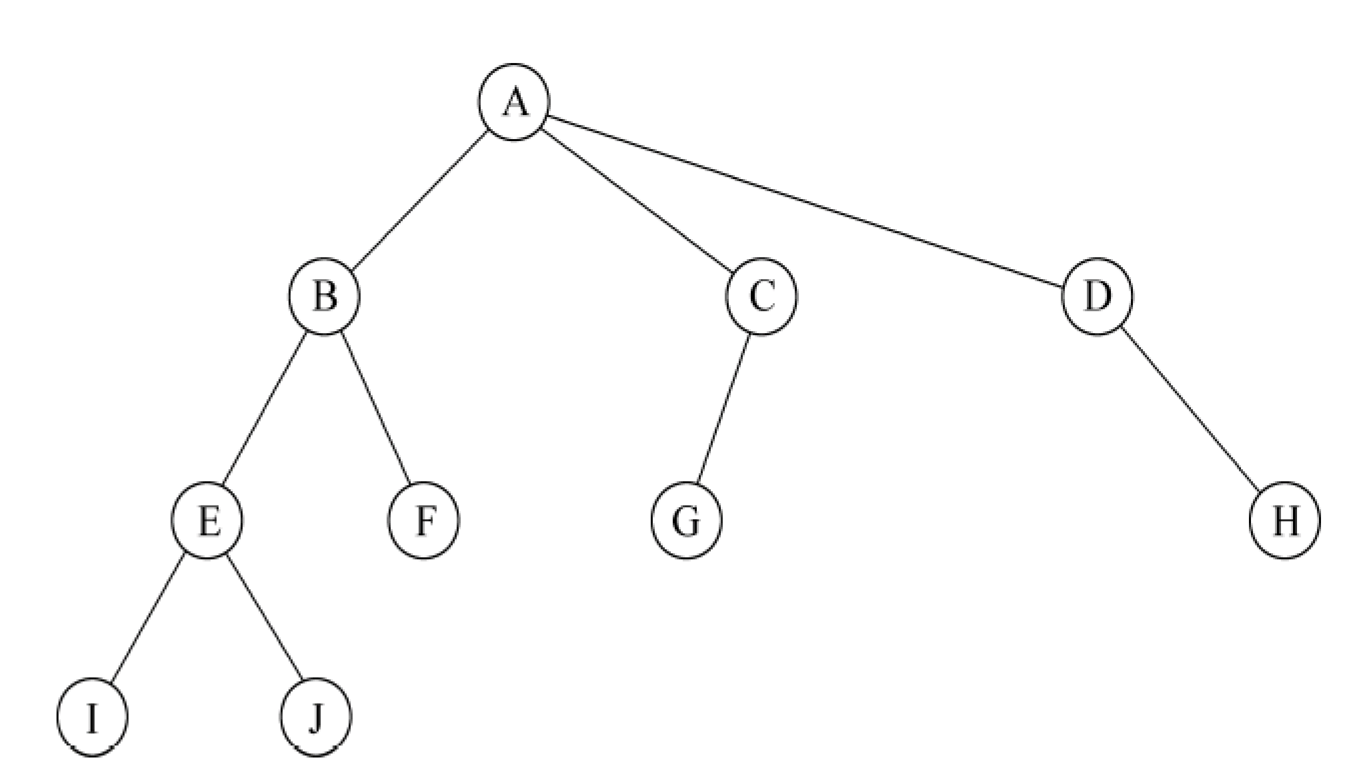


图 4.20 调用图

第5章 系统测试

系统测试是指将测试软件放到运行环境中,对该软件具体运行情况的测试。比如将该软件与硬件、外设和数据库等元素结合起来测试其综合性能。由于软件的含义广泛,所以在系统测试时,我们考虑的不单单是程序本身,这就需要在系统测试过程中制定系统测试的需求分析、系统总体设计以及详细设计等内容。此外,系统测试要求测试环境尽可能接近真实运行环境,否则设计的测试数据的参考价值就会受限。一般情况下,系统测试需要找专门的第三方软件公司来帮助完成。

总的来说,系统测试需要从16个方面来考虑,分别是功能测试、性能测试、压力测试、容量测试、安全性测试、GUI(图形用户界面)测试、可用性测试、安装测试、配置测试、异常测试、备份测试、健壮性测试、文档测试、在线帮助测试、网络测试和稳定性测试。

对于系统测试这16个方面谁先进行,谁后进行呢?严格来说,没有一个固定的顺序。但是一般都先进行功能测试,其次进行性能测试,之后再进行容错性测试、兼容性测试和系统安全性测试等。

对于功能测试,前面几章已经进行了详细的介绍,包括黑盒测试、白盒测试等不同的测试方法。本章主要分析几种重要的非功能性测试,分别是性能测试、压力测试、容量测试、可靠性测试以及GUI测试。

5.1 性能测试

性能测试(performance test)主要检验软件是否达到需求规格说明书中规定的各类性能指标,并满足一些性能相关的约束和限制条件。通过性能测试,确认软件是否满足产品的性能需求,同时发现系统中存在的性能瓶颈,以此对系统进行优化。

那么什么是性能呢?从用户角度来说,性能就是系统解决用户问题的时间。一般来说,性能是一种指标,表明软件或构件对于其及时性要求的符合程度;其次,性能是软件产品的一种特性,可以用时间来进行度量。

例如,"用户点击网站某个链接后3秒内链接内容展现出来","用户输入用户名、密码后,单击登录按钮,3秒内完成登录,进入系统首页面",这些都是用户对任务响应时间的描述。简单地说,软件性能反映的是一种响应速度,速度越快,可以说软件性能就越好;相反,如果一个软件用起来总是反应比较迟钝,一直处于等待响应状态,那就可以说这个软件性能比较差。

对于"性能",可以从3个不同角度来看。

(1) 用户的角度;

(2) 管理员的角度;

(3) 开发人员的角度。

从以上三种视角对性能测试都会有不同的理解和定义。

1. 用户角度

首先从用户角度来看,用户最关心的就是系统的响应时间,系统响应时间还要包括呈现时间,如图 5.1 所示。因此,在测试过程中要对具体响应时间有一个量化的数值体现。

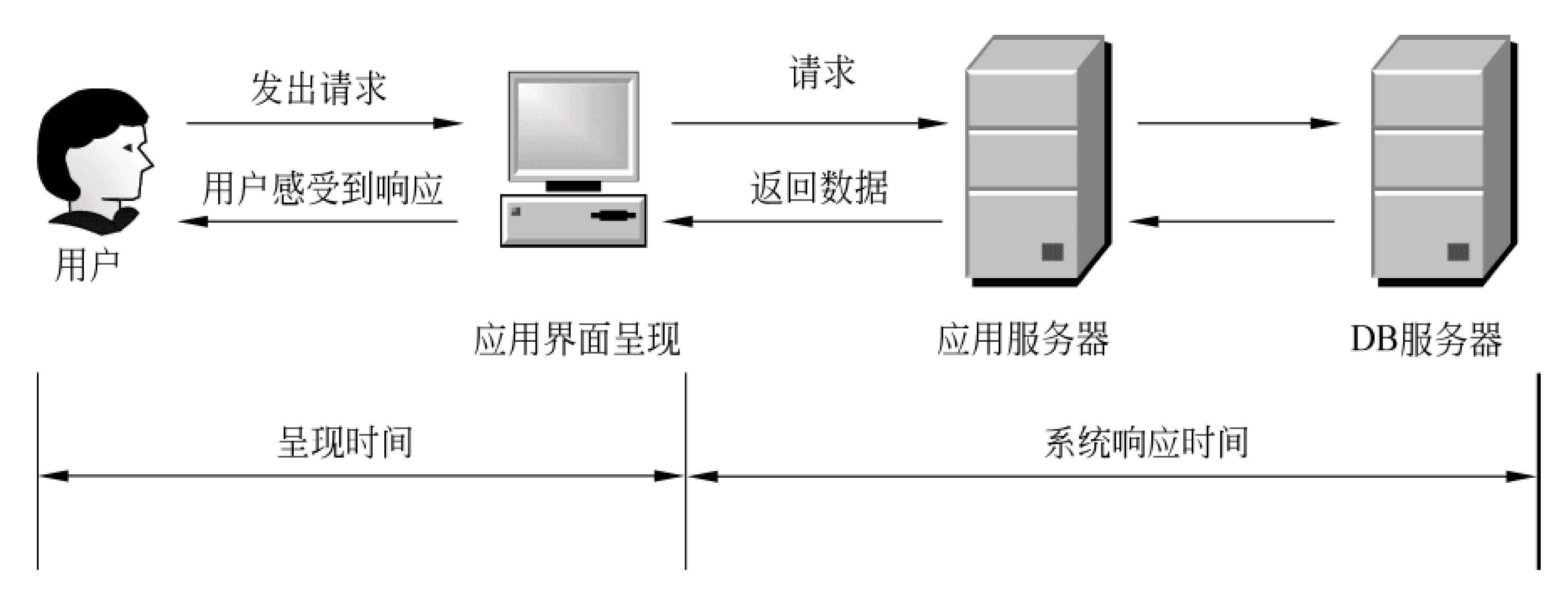


图 5.1 从用户角度看系统响应时间

2. 管理员角度

从管理员角度,性能测试又具有哪些内容呢?不同角色的管理员关心的性能是不同的。比如对于数据服务管理员,除了响应时间以外,合理的数据存取和服务器的资源使用等都是其要了解和掌握的内容。总的来说,测试人员需要给管理员这一角色提供的系统性能数据包括以下几个:

(1) 服务器的资源使用状况。

(2) 系统是否能够实现扩展。系统在使用过程中,随着客户需求和客户量的增加,要求扩展系统功能,那么测试人员应该能量化系统的可扩展能力。

(3) 系统最大的业务处理量。通过性能测试不同的方法,了解系统的瓶颈在哪里——是程序算法对系统性能进行了限制,还是硬件或网络对算法的性能进行了限制。

(4) 系统性能系统能否支持 7×24 小时的业务访问。

3. 开发人员角度

对于开发人员来说,需要测试人员提供他所关心的性能的测试结果,开发人员关心的是与程序开发相关的性能问题,也就是所说的代码问题。具体可以从以下几个方面来观察性能数据。

(1) 软件架构设计。

(2) 数据访问。不同的数据访问方式,可以实现不同的性能,比如 Select * from table1 where a<>1 和 Select * from table1 where a<1 or a>1 效率不同。

(3) 内存使用方式。不同数据定义形式的内存占用情况是不同的,比如静态分配形式和动态分配形式。当数据量较大时,内存占用方式在很大程度上影响了系统的运行性能。所以内存使用方式需要开发人员仔细考虑。

(4) 线程同步方式。有临界区、互斥区、事件和信号量 4 种方式,不同方式在实现线程同步时效率也是不同的。例如,临界区方式通过对多线程的串行化来访问公共资源或一段代码,速度较快。但是不同的程序设计需要不同的线程同步方式,这也需要开发人员仔细考虑。

系统测试中应提供系统评估能力。比如,测试中可以得到负载大小和响应时间等量化

的数据,而这些数据可以被用于验证计划好的模型,并帮助做出决策。

此外,在系统测试中要发现系统的弱点,找到系统瓶颈,受控的负荷可以被增加到一个极端的水平并突破它,从而修复系统的瓶颈或薄弱的地方。

最后,进行系统调优,重复运行测试,以验证调整系统的活动得到了预期的结果,从而改进性能,检测软件中的问题。

一般性能测试需要使用工具帮助完成,比如后续章节介绍的 LoadRunner,然而如何量化系统测试呢?以什么样的标准进行测试呢?在实际的测试过程中经常使用基准法进行衡量,常用的衡量标准如下。

1. 响应时间

响应时间是从应用系统发出请求开始,到客户端接收到最后一个字节数据为止所消耗的时间。合理的响应时间取决于实际的用户需求,不能根据测试人员的设想来决定。

例如,对于一个电子商务网站,在美国和欧洲,一个普遍被接受的响应时间标准为 2/5/10,也就是说,在 2 秒之内给客户响应被用户认为是"非常有吸引力的",在 5 秒之内响应客户被认为是"比较不错的",而 10 秒是客户能接受的响应的上限。但对于一个财务报账系统,该系统的用户每月使用一次该系统,一次花费 2 小时以上进行数据的录入,当用户单击"确认"按钮后,即使系统在 20 分钟后才给出"处理成功"的消息,用户仍然不会认为该系统的响应时间是不能接受的。

2. 并发用户数

并发用户数一般是指同一时间段内访问系统的用户数量。这里需要区分并发用户数与在线用户数和系统用户数的区别。并发用户是同一时间访问系统的用户,而系统用户为使用这个系统的所有用户,在线用户是登录系统的用户,而该用户可以不进行任何操作,例如仅仅浏览网页,不会对服务器造成任何负担。一般情况下有"系统用户数≥在线用户数≥并发用户数"这样的关系。要统计准确的并发用户数还是有一定难度的,一般都是估算并发用户数。例如,对于企业内部使用的 Web 系统,经常用每天访问系统的用户数(也就是在线用户数)的 10%作为并发用户数。当然并发用户数也可以由客户提供。

3. 吞吐量

吞吐量指单位时间内系统处理的客户请求数量,直接体现软件系统的性能承载能力。吞吐量一般都表示成 n 个请求/秒、n 个页面/秒、n 人/天或 n 个处理的业务/小时。可以根据吞吐量的变化来分析系统的瓶颈,比如,随着用户数的增加,吞吐量出现急剧的减少,那么可以确定出系统瓶颈数据。

4. 性能计数器

性能计数器是描述服务器或操作系统性能的一些数据指标,比如 Windows 系统的"任务管理器"中提供的"性能"数据,如图 5.2 所示。



图 5.2 Windows 系统的性能计数器

5.2 压力测试

压力测试(stress testing)是指模拟巨大的工作负荷,以检验系统在峰值使用情况下是否可以正常运行。

一般情况下,压力测试通过逐步增加系统负荷来测试系统性能的变化,并最终确定在什么负荷条件下系统性能处于失效状态,以此来获得系统性能提供的最大服务级别。压力测试经常用来测试系统的稳定性,通过测试工具使用模拟的方法检查系统在受压力的情况下正常运行的能力。

压力测试不同于性能测试,是用来保证产品发布后系统能否满足用户需求。因此压力测试更多地关注系统的整体。而性能测试可以针对局部进行,比如对单独的一个模块也可以进行性能测试,但是不能对局部进行压力测试。

此外,压力测试关注的是系统在巨大工作负荷下的工作状况,而性能测试中系统一般工作于正常负荷下。所以压力测试是系统在峰值处的性能状况,经常用来测试系统的稳定性。

例如,对一个系统进行测试,模拟并发用户数为50~100以观测系统表现,就是在进行常规的性能测试;当用户增加到系统出现性能瓶颈时,如1000乃至上万个用户时,就变成了压力测试。

那么如何进行压力测试呢? 一般情况下,压力测试是通过增加负载的方式来进行的,比如并发用户数的增加、系统资源的减少等。压力测试可以通过一次性加载、递增加载、突变加载和随机加载等方式实现,如图5.3所示。

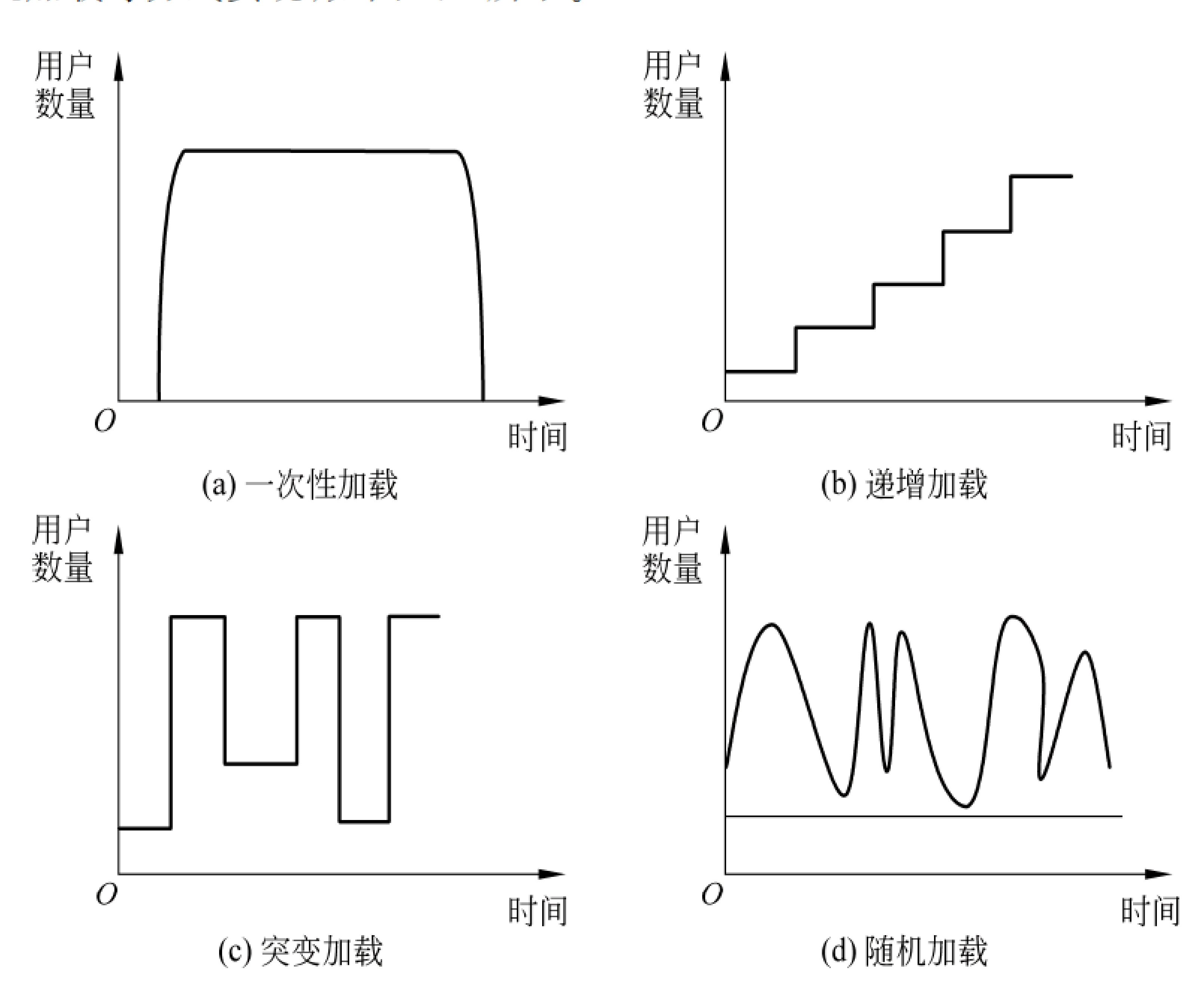


图5.3 压力测试方法

通过加载可以实现压力测试。在压力测试过程中,应根据系统不同的侧重点,尽可能逼真地模拟系统环境。比如在实时系统中,要以正常或者超常的速度进行测试。另外,在批处

理系统中，可以利用更多的任务进行多任务测试。

具体来说，针对实时系统进行压力测试，测试人员应使用标准文档，模仿系统产品化之后的用户，对系统时间进行扩展，在扩展后的时间段内增加负载。针对批处理系统，测试人员至少要应用多于一个事务的批量进行压力测试。

一般在测试过程中要采用多种测试方法，具体可以归纳为三类压力测试方法，分别为重复、并发以及增大量级。

1. 重复测试

重复测试就是一遍又一遍地执行某个操作或功能，比如重复调用一个 Web 服务。压力测试的一项任务就是确定在极端情况下一个操作能否正常执行，并且能否持续不断地在每次执行时都正常。这对于推断一个产品是否适用于某种使用情况至关重要，客户通常会重复使用产品。重复测试往往与其他测试手段一并使用。

2. 并发测试

并发是同时执行多个操作的行为，即在同一时间执行多个测试线程。例如，在同一个服务器上同时调用许多 Web 服务。并发测试不一定适用于所有产品，但多数软件都具有某些并发行为或多线程行为元素，这一点只能通过执行多个代码测试用例才能得到测试结果。

3. 量级增加

压力测试可以重复执行一个操作，而操作自身也要尽量给产品增加负荷。例如，对于一个局域网通信业务，要检测从发送方发出消息到接收方收到，测试人员可以通过模拟输入超长消息来使操作强度提高，即增加这个操作的负荷量级。这个负荷量级的确定总是与应用系统有关，可以通过查找产品的可配置参数来确定负荷量级。

压力测试是保证系统稳定性的关键测试技术，然而压力测试对系统资源要求较高，所以也是耗资很大的一种测试手段。

5.3 容量测试

容量测试的主要任务是测试系统承载量级的变化。与压力测试的概念很相近，然而两者的侧重点不同。

在进行容量测试时，测试人员需要用一些特定的手段，比如制造特定量级的任务，达到系统极限。测试人员在进行容量测试的过程中，可以发现系统承受超额的数据容量的极限状态，以及在这一状态下系统是否能够正确处理任务，从而分析出系统应用特征的某项指标的极限值，如最大并发用户数、数据库记录数等。

容量测试和压力测试很相近，容量测试是采用特定的手段测试系统能够承载处理任务的极限值。然而容量测试更关注于容量和量级变化。容量测试会让系统承受超额的数据容量，通过测试反映出系统某一个性能的变化。

那么如何进行容量测试，有什么常用的方法吗？通常，容量测试的目的是确定系统数据量级上的临界值或极限。测试人员根据实际运行中可能出现的极限，制造相应的任务组合，以此激发系统出现极限。通常，测试人员可以从以下几个方面来考虑：

(1) 数据库表的数量；

(2) 数据量的大小；

(3) 文件的大小和数量;

(4) 数据输入的量值。

在测试过程中要注意系统在临界情况下出现的问题。比如磁盘数据的丢失、缓冲区溢出、数据丢失以及系统整体性能下降等。在进行容量测试时,还需要注意数据生成的过程,可以利用工具随机生成所需数据,或者使用数据的复制。在系统运行时,存储碎片对数据存储和系统调用都存在不同程度的影响,测试人员应将其考虑在内。

简单地说,容量测试需要检查当系统运行在大量数据甚至最大或更多的数据测试环境下是否会出问题。容量测试是必要的。那么在什么情况下,才能体现出容量测试的优势呢?一般情况下,在软件平时的使用过程中,数据容量的变化存在使用高峰和低谷。例如报税系统,在月底和年底的时候会有较大的数据量,而在平时,数据量并不是很大。在这种情况下,容量测试是很必要的。在容量测试过程中,硬盘、数据库、文件夹、缓冲区以及数据通信过程中的数据容量都应引起测试人员关注。

另外,不容忽视的是,在容量测试过程中,还可以考虑容量最小情况下系统的可用性以及稳定性,相应存在的是空的数据库、空的数据表、空邮箱、空链接以及长时间无操作状态等。

下面利用不同的场景,举例分析容量测试的过程。

(1) 输入数据模块。

测试分析:

① 数据输入速度快;

② 数据内容长;

③ 数据包含特殊字符。

测试结果:缓冲区是否溢出,是否被填满。

(2) 访问数据库。

测试分析:

① 混合使用创建、更新、读和删除等不同的操作;

② 数据库包含最大的容量;

③ 建立访问数据库对象的最大数量的实例。

测试结果:访问数据库时间和正确性。

(3) 文件管理。

测试分析:

① 文件大小;

② 服务器磁盘空间满;

③ 文件交换。

测试结果:文件过大或者磁盘空间满时是否有警告,是否存在数据丢失等不合理现象。

(4) 文件交换时的通信过程。

测试分析:

① 通信路径是否正确;

② 通信过程文件是否丢失;

③ 通信过程文件是否重复;

④ 所占用的时间是否可以接受。

测试结果：通信过程是否正确。记录通信时间。

压力测试、容量测试和性能测试的测试方法相通，在实际测试工作中，往往结合起来进行以提高测试效率。一般会设置专门的性能测试实验室完成这些工作，即使用虚拟的手段模拟实际操作，所需要的客户端有时还会很大。因此性能测试实验室的投资往往会很大。对于许多中小型软件公司，可以委托第三方完成性能测试，可以在很大程度上降低成本。

5.4 可靠性测试

可靠性测试是一项很重要的测试，不同于压力测试和容量测试，一般情况下，可靠性测试会提供一个量化的、统一的指标。可靠性测试一直是学者和专家非常关注的领域，在硬件可靠性界定方面，不同领域都存在一套较为合理的量化指标。随着软件应用的发展，尤其是越来越智能的软件系统给人们的工作和生活带来了巨大的变化。软件的风险以及软件的安全性和可靠性也同时受到了重视。但是，智能软件可靠性的界定确实也存在着较大的困难。

然而什么是软件可靠性呢？美国电气和电子工程师协会（IEEE）计算机学会在1983年对“软件可靠性”作了明确的定义：在规定的条件下和时间内，软件不引起系统失效的概率。其中所说的概率是系统输入和系统使用的函数，也就是软件中存在的故障的函数，系统输入将确定是否会触发已存在的故障（如果故障存在）。我国在1989年将该定义接受为国家标准。该定义主要有两方面的含义：

第一，在规定的条件下和规定的时间内，软件不引起系统失效的概率。具体来讲，规定的条件是指直接与软件运行相关的使用该软件的计算机系统的状态和软件的输入条件，或统称为软件运行时的外部输入条件。例如，软件的环境条件包括软件运行、存储等相关的内外部软硬环境。这些因素对软件的正常运行都有着非常重要的影响。规定的时间也对软件的可靠性有重要的影响，在不同的时间内，软件表现出的可靠性不同。

第二，在规定的时间周期内，软件为了提供给定的服务所必须具备的功能。

在这个定义中说明了两个关键的可靠性因素，一个是规定的时间和规定的条件。因为我们软件开发都是为了履行某一个特定的功能。在软件需求定义时，需要明确软件的工作条件、系统的使用环境以及规定的工作时间等。如果在测试过程中，软件达不到需求说明书的要求，就称之为“失效”，有时候也称之为“故障”。一般情况下，称之为“失效”，这是因为“故障”经常指不可修复或不需修复的问题。另外，系统工作时间也是一个广义的定义，根据实际情况，这个“时间”的范围可以扩展到“周期”、“次数”或“瞬间”等。

5.4.1 可靠性度量

可靠性是一个重要的软件衡量标准。计算软件的可靠性不是个简单的事情，不同的运行剖面，不同的测试用例，会得到不同的结果。一般情况下，可以使用本节介绍的几个概念来进行可靠性度量。

为了量化可靠性，首先假定软件正常工作和维护的场景如下（见图5.4）：系统投入使用，工作了一段时间 t_1 后，出现一个故障，需要进行系统维护，假设维护时间为 T_1；故障清除后，系统继续投入使用，正常工作的时间为 t_2，又出现故障，维护此故障的时间为 T_2；与此过

程相类似，统计 n 次工作时间和维护的时间，也就是参数$\{t_1, t_2, \cdots, t_n\}$和$\{T_1, T_2, \cdots, T_n\}$。

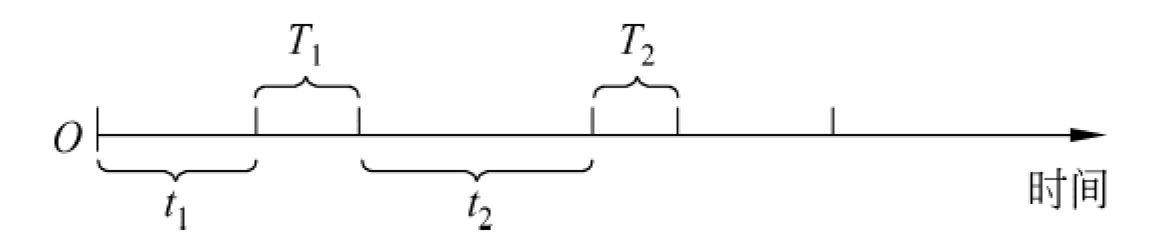


图 5.4　系统工作和维护时间

1. 故障率（风险函数）

故障率 λ 表示单位时间内发生失效的次数，其计算公式如下：

$$\lambda = \frac{\text{总失效次数}}{\text{总工作时间}} = \frac{n}{\sum_{i=1}^{n} t_n}$$

一般 λ 取 10^9 个单位时间内发生的故障数。加入测试模块数为 m，测试时间为 t，那么在 $m \times t = 10^9$ 的时间内发生的故障数就是故障率。

2. 维修率

维修率表示在单位维护时间内发生失效的次数。其计算公式如下：

$$\mu = \frac{\text{总失效次数}}{\text{总维护时间}} = \frac{n}{\sum_{i=1}^{n} T_i}$$

3. 平均无故障工作时间

有了故障率和维修率后，平均无故障工作时间和平均维护时间都可以从这两个基本参数中推导出来。

平均无故障工作时间 MTBF(Mean Time Between Failures)表示连续正常工作的时间的平均值，其计算公式如下：

$$\text{MTBF} = \frac{\text{总工作时间}}{\text{总失效次数}} = \frac{\sum_{i=1}^{n} t_i}{n} = \frac{1}{\lambda}$$

4. 平均维护时间

与平均无故障工作时间类似，平均维护时间 MTTR(Mean Time To Repair)指系统在发生多次故障时，对维护时间的统计并求平均，也就是平均故障时间。这个值可以较好地说明系统的可维护性，MTTR 小，说明系统可维护性相对较好。其计算公式如下：

$$\text{MTTR} = \frac{\text{总维护时间}}{\text{总失效次数}} = \frac{\sum_{i=1}^{n} T_i}{n} = \frac{1}{\mu}$$

5. 有效度

有效度表示系统在某个时间单位内正常工作的概率。如果有效度高，则系统可工作时间较大。其计算公式如下：

$$A = \frac{\text{总工作时间}}{\text{总工作时间} + \text{总维护时间}} = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}} = \frac{\mu}{\mu + \lambda}$$

6. 可靠性

可靠性指系统运行多次不发生故障的概率。可靠性可表示为

$$R(n) = P$$

其中 n 表示系统运行次数。

假设系统运行 n 次出现故障的概率为 $Q(n)$。

在 t 时刻，系统的可靠性为 $R(t)$，同时，系统故障概率函数为 $Q(t)$。如果 T 为系统从开始工作到首次出现故障的时间，那么系统的可靠性 $R(t)$ 也就是系统运行到 t 时刻不出现故障的概率。那么，$R(t)=1-Q(t)$；同时 $R(t)=P(T>t)$。

这里，假设 $\lambda(t)$ 为 t 时刻的故障率，即在 t 时刻，单位时间 Δt 内出现故障的概率。下面分析 $R(t+\Delta t)$，

$$R(t+\Delta t)=P(T>(t+\Delta t))=P((T>t)\wedge(t,\Delta t))$$
$$=P(T>t)+(1-\lambda(t)\Delta t)=R(t)+(1-\lambda(t)\Delta t)$$

因此，

$$\mathrm{d}R=-\lambda(t)R(t)\mathrm{d}t$$

计算得出可靠性：

$$R(t)=\mathrm{e}^{-\int_0^t \lambda(x)\mathrm{d}x}$$

本节介绍的 6 个与可靠性相关的专业术语在软件可靠性分析过程中是经常用到的。一般情况下，在软件开发过程中，首先需要统计在有限次数内软件的故障率 λ 以及维修率 μ。根据 λ 以及 μ 分析系统目前运行的稳定性和可靠性。如果不能满足需求，需要进行其他的操作，比如重新优化软件。如果结果可以满足需求，利用一些经典的、合理的模型来估计系统在将来的运行过程中的可靠性，并估计下一个时间段软件无故障工作时间、故障发生的时间以及维护时间等。到目前为止已经有上百种软件可靠性模型，在 5.4.2 节中会详细介绍 3 种典型的、有代表性的软件可靠性模型。

5.4.2 可靠性模型

软件可靠性模型自 1972 年由 Z. Jelinski 和 P. Moranda 提出 J-M 模型后，至今已经有几百种模型公开发表。软件可靠性模型通过分析软件目前已经存在的失效数据，进行建模，估计软件的可靠性并进行预测。由于 5.4.1 节可靠性计算过程中介绍的软件运行时间 t 是一个随机量，可以对其进行相应的分析，相关的专家根据随机过程、机器学习、模糊理论和混沌理论等内容进行可靠性模型的建立。

对于软件模型来说，人们借助于它要做的主要是两项工作：估算和预测。

(1) 估算：将统计推论过程作用于系统的失效数据。

(2) 预测：根据产品和开发过程，在程序执行之前就可以得到这些参数的值。

本节介绍 3 个经典的软件可靠性模型。

1. J-M 模型

在 J-M 模型中，假设一个故障一旦被发现，立即排除且不引入新的故障。另外，J-M 模型还假设故障风险率在每次的排错过程中由一个常量变为另一个常量，但在两次排错之间不变，即故障风险率是分段常数。J-M 模型采用最大似然法估计出软件中的总故障数、残留故障数与风险率之间的比例系数。

J-M 模型中存在以下假设：

(1) 程序中的固有故障数 N_0 是一个未知的常数。

(2) 程序中的各个故障是相互独立的，每个故障导致系统发生失效的可能性大致相同，

各次失效间隔时间(即故障发生间隔时间相同)也相互独立。

(3) 测试中检测到的故障都被排除,每次排错只排除一个故障,排除时间可以忽略不计,在排错过程中不引入新的故障。

(4) 程序测试环境与预期的使用环境相同。

(5) 程序的失效率在每个失效间隔时间内是常数,其数值正比于程序中残留的故障数,在第 i 个测试区间,其失效率函数为 $\lambda(x_i)=\phi(N_0-i+1)$,其中 ϕ 表示其比例常数,而 x_i 为第 i 次失效间隔中以 $i-1$ 次失效为起点的时间变量。

J-M 模型认为,第 $i-1$ 次失效为起点的第 i 次失效发生的时间是一个随机标量,它服从以 $\phi(N_0-i+1)$ 为参数的指数分布,所以密度函数表示为

$$P(x_i)=\phi(N_0-i+1)\exp\{-\phi(N_0-i+1)x_i\}$$

也就是说 x_i 的分布函数为

$$F(x_i)=1-\exp\{-\phi(N_0-i+1)x_i\}$$

可靠性函数为

$$R(x_i)=1-F(x_i)=\exp\{-\phi(N_0-i+1)x_i\}$$

这里存在两个变量 N_0 以及 ϕ。可以利用参数估计的方法来计算得到两个变量的估算值。一般情况下,J-M 算法利用的是最大似然函数估算法来对变量进行计算。这里不再详细介绍求解过程。

$$\sum_{i=1}^{n}\frac{1}{\hat{N}_0-i+1}=\frac{n}{\hat{N}_0-\left(\sum_{i=1}^{n}x_i\right)^{-1}\sum_{i=1}^{n}(i-1)x_i}$$

上式超越方程组的求解可以得到 N_0 的估算值。有了 N_0,就更方便地得到 ϕ。

$$\hat{\phi}=\frac{n}{\hat{N}_0\sum_{i=1}^{n}x_i-\sum_{i=1}^{n}(i-1)x_i}$$

2. G-O 模型

G-O 模型是非齐次泊松过程模型,它是在 1979 年由 A. L. Goel 和 K. Okumoto 提出的。一般称之为 NHPP 类 G-O 模型,也就是非齐次泊松过程模型。G-O 模型利用的是 NHPP 描述的软件错误查出模型,或称为查出指数类增长模型。

G-O 模型提出以下假设:

(1) 软件在与预期的操作环境相似的条件下运行。

(2) 在任何时间间隔内检测到的故障数是相互独立的。

(3) 每个故障的严重性和被检测到的可能性大致相同。

(4) 在 t 时刻检测出的累积故障数 $N(t)(t\geqslant 0)$ 是一个独立增量过程,$N(t)$ 服从期望函数 $m(t)$ 的泊松分布,在 $(t,t+\Delta t)$ 时间区间中发现的故障数的期望值正比于 t 时刻剩余故障的期望值。

(5) 其中,累积故障数的期望函数 $m(t)$ 是一个有界的单调增函数,并满足:

$$\begin{cases}m(0)=0\\ \lim\limits_{t\to\infty}m(t)=a\end{cases}$$

式中,a 为最终可能被检测出的失效总数的期望值。

由于在$(t,t+\Delta t)$时间区间中发现的故障数的期望值正比于t时刻剩余故障的期望值，因此有

$$m(t,t+\Delta t)-m(t)=b(a-m(t))+O(\Delta t)$$

其中，$O(\Delta t)$表示Δt的高阶无穷小。

$$b=\frac{m(t,t+\Delta t)-m(t)}{(a-m(t))\cdot\Delta t}$$

即

$$m'(t)+bm(t)-ab=0$$

可以得到

$$m(t)=a(1-\mathrm{e}^{-bt})$$

又因为$N(t)$服从期望函数$m(t)$的泊松分布，因此有

$$P\{N(t)=k\}=\frac{(m(t))^k}{k!}\mathrm{e}^{-m(t)}$$

于是剩余的故障总数$\overline{N}(t)$

$$E(\overline{N}(t))=a-m(t)=a\mathrm{e}^{-bt}$$

软件发生第n次失效后，可靠性函数为

$$\begin{aligned}R_{n+1}(x\mid T_n=s)&=P\{X_{n+1}>x\mid T_n=s\}=\exp\{-a[\mathrm{e}^{-bs}-\mathrm{e}^{-b(s+x)}]\}\\&=\exp\{m(x)\times\mathrm{e}^{-bs}\}\end{aligned}$$

假定软件第n次失效发生在s时刻，此时MTTF_{n+1}为

$$\mathrm{MTTF}_{n+1}=\int_0^{\infty}R_{n+1}(x\mid T_n=s)\mathrm{d}x=\int_0^{\infty}\exp[-m(x)\times\mathrm{e}^{-bs}]\mathrm{d}x$$

可以利用最大似然估算法计算a、b的值。

3. Trividi 模型

在分析前两个模型时，将失效的发生看作是一个随机的变量来进行建模，而没有考虑到变量之间的相关性。Trividi 模型分析了失效的相关性，将软件运行的成功与失败看作贝努里分布(见图5.5)，对每次的运行结果组成的序列采用离散马尔可夫链进行建模。

Trividi 模型提出以下假设：

(1) 当前软件运行的结果取决于上次软件运行结果。

(2) 当一个失效发生，相应故障马上被查出且修正。

(3) 每次失效发生后，软件下次运行成功或失败的概率改变。

(4) 在测试阶段，软件每次运行时间的分布不会改变。

根据以上的假设，用Z表示软件运行成功或失败：

$Z=0$ 表示软件运行成功；

$Z=1$ 表示软件运行失败。

运行结果序列构成了一个离散马尔可夫链，如图5.6所示。底线上圆圈中的数字代表查到第i次失效。第i次失效之后，下次失效之前，会有几次成功运行，图中的i_s代表这种状态。当然，一次软件失效之后，下次运行也可以是失效，因而从状态i到状态$i+1$有一条连线。

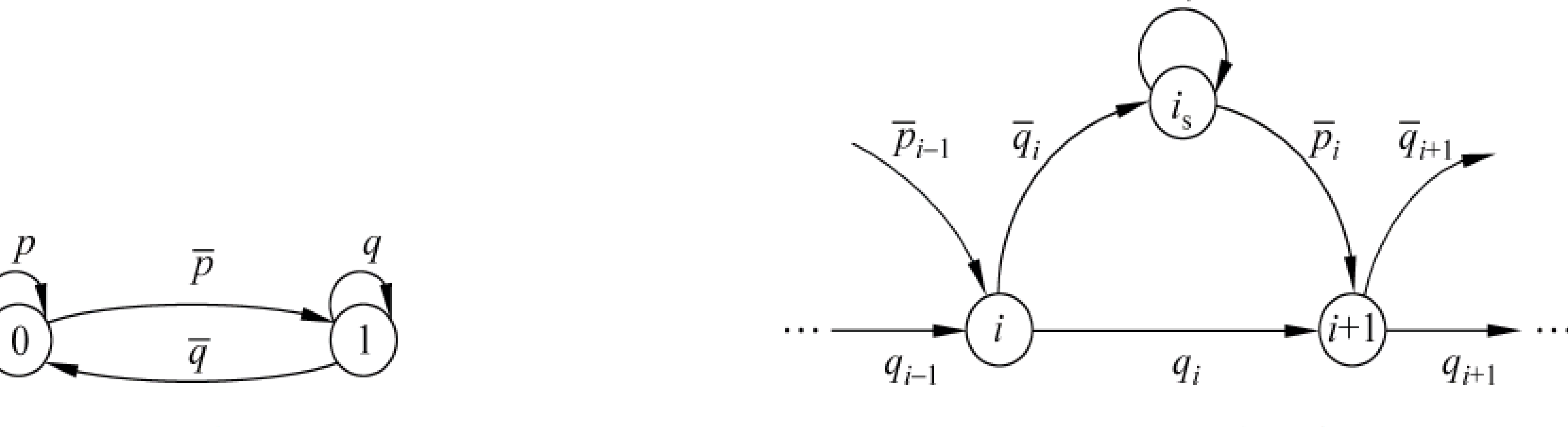


图 5.5　贝努里分布　　　　图 5.6　运行结果序列

当软件失效是非独立的情况下，每次运行结果的概率取决于上次的运行结果：

$$q_i = P(Z_{i+1} = 1 \mid Z_i = 1)$$
$$\bar{q}_i = P(Z_{i+1} = 0 \mid Z_i = 1)$$
$$p_i = P(Z_{i+1} = 0 \mid Z_i = 0)$$
$$\bar{p}_i = P(Z_{i+1} = 1 \mid Z_i = 0)$$

在图 5.6 中，从一个状态转移到另一个状态需要一定的时间。定义 $F_{ij}(t)$为从状态 i 到状态 j 所需要的时间的分布函数。可以从运行结果来分为 $F_{exS}(t)$与 $F_{exF}(t)$，分别表示运行成功的时间分布函数与运行失败的时间分布函数。

用 x_{i+1}表示软件在第 i 次和第 $i+1$ 次失效之间的运行次数。x_{i+1}的分布律为

$$P(x_{i+1} = k) = \begin{cases} q_i, & k = 1 \\ \bar{q}_i p_i^{(k-2)} \bar{p}_i, & k \geqslant 2 \end{cases}$$

在系统经历 i 次失败的条件下，第 $i+1$ 次失败的时间的条件分布函数为

$$F_{i+1}(t) = q_i F_{exF}(t) + \sum \bar{q}_i p_i^{(k-2)} \bar{p}_i F_{exS}^{(k-1)^*}(t) F_{exF}(t)$$

其中，* 表示卷积，$F_{exS}^{(k-1)^*}(t)$表示 $F_{exS}(t)$的 $k-1$ 次卷积。

两边同时进行 LST 变换，可以得到

$$\widetilde{F}_{i+1}(s) = q_i \widetilde{F}_{exF}(s) + \sum_{k=2}^{\infty} \bar{q}_i p_i^{k-2} \bar{p}_i \widetilde{F}_{exS}^{(k-1)}(s) \widetilde{F}_{exF}(s)$$

当软件运行时间服从指数分布时，有

$$F_{exS}(t) = 1 - e^{-\mu_S t}, \quad F_{exF}(t) = 1 - e^{-\mu_F t}$$

其中 μ_S 和 μ_F 均为正的常数。

对上式同样进行 LST 变换：

$$\widetilde{F}_{exS}(s) = \frac{\mu_S}{s + \mu_S}, \quad \widetilde{F}_{exF}(s) = \frac{\mu_F}{s + \mu_F}$$

结合以上两个 LST 变换后的公式可以得到

$$F_{i+1}(t) = \frac{q_i \mu_F - \bar{p}_i \mu_S}{\mu_F - p_i \mu_S}(1 - e^{-\mu_F t}) + \frac{\bar{q}_i \mu_F}{\mu_F - p_i \mu_S}(1 - e^{-\bar{p}_i \mu_S t})$$

这样可靠性就可以计算出来：

$$R(t) = 1 - F_{n+1}(t)$$

5.4.3　软件运行剖面

所谓软件的运行剖面，是用来描述软件的实际使用情况的一个概念。

首先对软件的使用者进行分类，不同类型的使用者可能以不同的方式来使用软件，根据对使用者的划分将软件划分成不同的模式剖面。

另外，模式剖面又可以划分为不同的功能剖面，即每个模式下都有许多不同的功能。最后，每个功能又由许多运行组成，这些运行的集合便构成了运行剖面。

以上是软件运行剖面的一个基本概念，如何对它进行定量的分析描述呢？

定义 5.1 软件的运行剖面。设 D 是软件 S 的定义域，$D=\{d_1,d_2,\cdots,d_n\}$，$P(d_i)$是 d_i 的发生概率，则运行剖面被定义为$\{(d_1,P(d_1)),(d_2,P(d_2)),\cdots,(d_n,P(d_n))\}$。

如果$\forall d\in D$，若 $S(d)$是正确的，那么 S 正确运行的概率为 1，即 $P(S)=1$。假设 $D=D_1\cup D_2$，D_1 是 S 正确运行的概率，D_2 是 S 产生错误的概率。在均匀分布情况下，S 正确运行的概率为

$$P(S)=\frac{\| D_1 \|}{\| D \|}$$

根据定义可以看出，软件的运行剖面是针对系统使用条件来定义的。也就是系统的输入值用其按时间的分布或按其在可能输入范围内的出现概率的分布来定义。运行剖面是否能代表软件的实际使用呢？这取决于可靠性工程人员对软件的系统模式、功能、任务需求及相应的输入激励的分析，同时也取决于他们对用户使用这些系统模式、功能和任务的概率的了解。运行剖面的构造质量将对测试、分析的结果是否可信产生直接的影响。

如何划分软件的运行剖面呢？对软件的使用者进行分类，不同类型的使用者可能以不同的方式来使用软件，根据对使用者的划分将软件划分成不同的模式剖面。如上所说，模式剖面又可以划分为不同的功能剖面，即每个模式下都有许多不同的功能，而每个功能又由许多运行组成，这些运行的集合便构成了运行剖面，如图 5.7 所示。

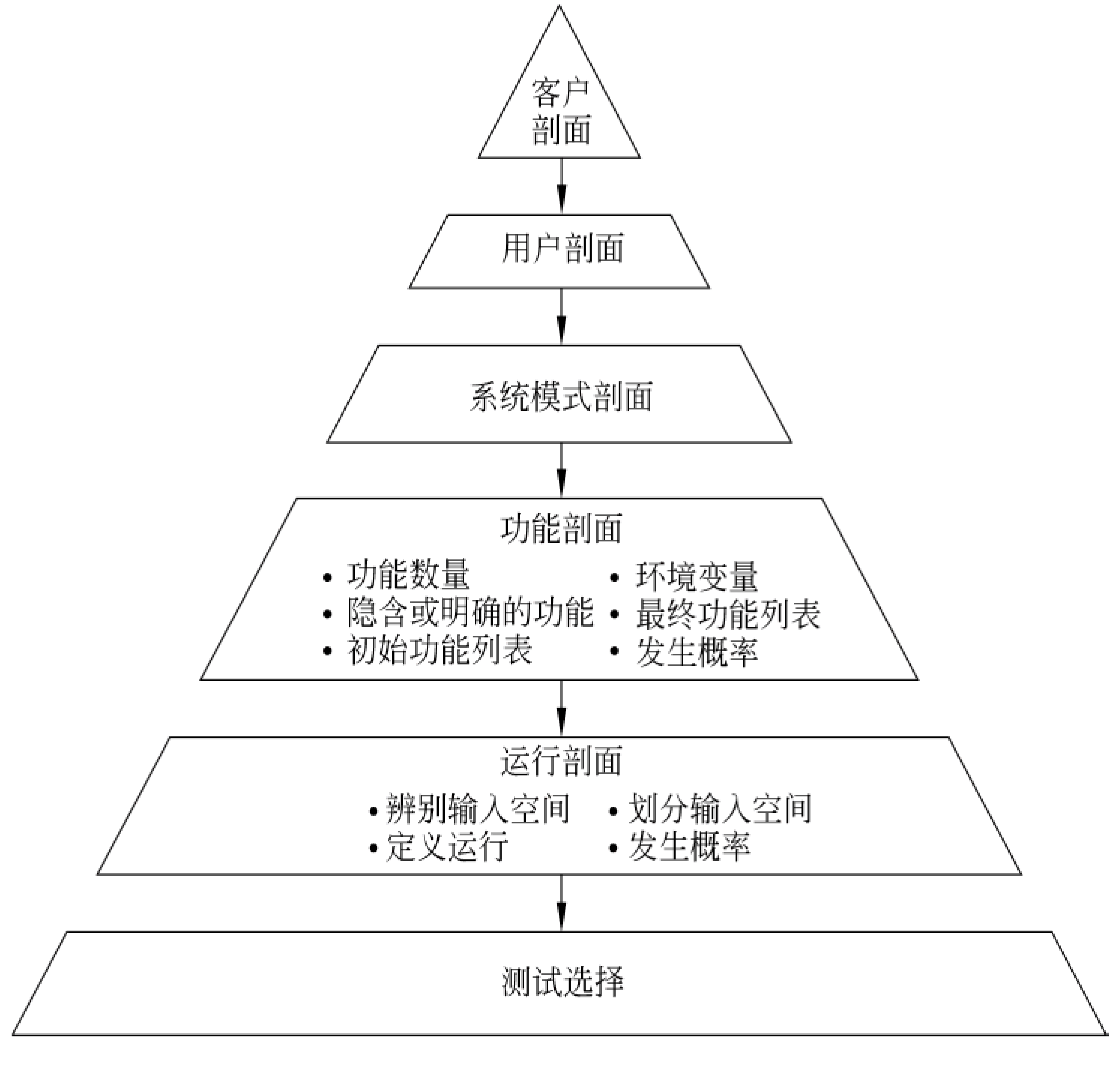


图 5.7 软件运行剖面

一般情况下，软件运行剖面是通过以上这些步骤分解而获取，即客户剖面→用户剖面→系统模式剖面→功能剖面→运行剖面。

客户剖面：以同一种方式使用系统的用户。客户剖面是客户及其发生概率的集合。一般情况下，客户剖面由不同的客户类型序列构成。比如在图书管理系统中，读者与管理员为两类不同类型的客户。

用户剖面：以同一种方法使用系统的用户称为一个用户组，不同客户的用户组可以以相同或不同的方式使用系统。用户剖面是在客户剖面基础上产生的。仍以图书管理系统为例，读者客户中有不同级别的读者：VIP、SVIP 等用户类别。

系统模式剖面：系统模式是为了便于管理系统或分析执行行为，分组而成的一个功能或操作的集合。系统模式可以来源于需求分析。

功能剖面：每种模式根据其任务书和运行环境，进一步分解为对应的功能，并确定每个功能发生的概率，构成功能剖面。

运行剖面：系统的具体实现功能，每一个功能由一系列运行组成。在运行剖面中需要确定这些运行发生的概率的集合。

5.5 GUI 测试

GUI(Graphics User Interface)即图形用户界面，因为 GUI 需要一些可复用的构件，所以在设计用户界面时更加省时而且更加准确，这也使得 GUI 测试越来越被重视。然而 GUI 测试涉及人机工程学、认知心理学、美学和色彩理论等不同方面的知识。

那么什么是 GUI 呢？用户界面设计从工作内容上来说分为 3 个方向：图形设计、交互设计和用户研究。用户界面经历了批处理、联机终端(命令接口)、(文本)菜单等多通道—多媒体用户界面和虚拟现实系统。用户界面中信息载体有以文本为主的字符用户界面、以二维图形为主的图形用户界面和多媒体用户界面，而用户界面的信息输出有以符号为主的字符命令语言、以视觉感知为主的图形用户界面、兼顾听觉感知的多媒体用户界面和综合运用多种感观(包括触觉等)的虚拟现实系统等不同的形式。

人机交互和计算机用户界面刚刚走过基于字符方式的命令语言式界面，目前正处于图形用户界面时代。但是，计算机科学家并不满足于这种现状，他们正积极探索新型的人机交互技术，比如多通道、虚拟显示技术、语音识别、体感知等不同的方式来进行人机交互。这就为用户界面测试带来了新的问题和发展动力。

然而，目前最流行的还是利用图形来与用户交流，利用计算机的图形能力产生的程序界面使得程序更加容易使用。一般来说，应用程序有以下基本组件：图标(icon)、窗口(window)、表单(Form)、属性页(Property Sheets)、菜单(menu)、工具栏(tool bar)、状态栏(Status bar)、进度栏(progress bar)、按钮(button)、对话框(dialog Box)、消息框(message Box)、输入对话框(input box)、文本框(Text Box)、列表框(List Box)、组合框(combo Box)、下拉列表框(Drop-down List Box)、复选框(Check Box)、单选框(Radio box)、选项框(Option box)、滑动条(Slider)、旋转按钮(Spin Button)、静态文字(Static tex)、向导(Wizards)和树(Tree)。

GUI 测试不论对于 Web 软件还是游戏类或者是应用类软件都是很重要的一项测试内容。其中第 3 章介绍的黑盒测试在 GUI 测试上做了很大的一部分工作。总的来说，进行 GUI 测试主要从以下几个方面来考虑：

（1）功能性；

（2）易用性；

（3）美观性；

（4）用户满意度。

下面详细分析这几个方面。

1. 功能性

功能性主要反映GUI在需求功能方面是否满足用户的需求。比如是否存在功能遗漏和冗余功能等。功能性的测试问题属于级别比较高的测试问题，在测试出来后，需要设计人员进行更正。

1）过度功能性

将简单功能复杂化，这是设计上一个较常见的问题。尝试进行太多工作任务的系统将很难学习和掌握，而且容易忘记。它要求大量的文档（开发文档、帮助文档和屏幕）。如果功能模块间过于紧密，则发生关联错误的几率要提高不少。有时候，用户需要的只是简单功能，而不要让它过于膨胀，成为一个“怪物”。

2）夸大的功能性印象

用户手册和营销传单不能使程序功能实现得更多，尤其是营销传单。记住，在用户手册中宁愿对功能略微轻描淡写也不能夸大其词。

3）对手头任务理解不适当性

可以把这个问题直观地理解为需求设计错误。对于任何一个项目，由于功能关键事项不存在、太有限或者太慢（需要改进程序结构或内部算法）而不能完成真正的工作。

举例来说，查询一个有8000条记录的数据库需要1个小时，虽然说具备了查询的功能，但是实在很令人怀疑此项功能是否会有人使用，更糟糕的情况是：用户由于使用了该功能而造成的恶劣印象难以在短时间内消除，虽然对于开发人员来说问题可能只是一个参数拼写错误而已。

4）遗漏功能

功能没有实现，却出现在了用户手册中；或者是本来应该具备的特征性功能，在程序中只能看到一个“影子”。出现这个问题多半情况下是由于需求变更时没有对手册进行检查和更新，也有可能是因为遗漏了需求说明中应包含的功能。

5）错误功能

一个本来应该完成查询工作的功能却干了排序的活儿。这种疏忽一般不是因为没有实现功能，而是在分配功能的时候出现了问题，当然这种情况的发现和排除应该不是一件困难的事。

6）功能性必须由用户创建

最常见的情况之一就是要求用户自己配置软环境（如配置数据源，一般都可以在安装程序中自动完成；当然还包括程序用到的组件在系统中不存在，安装程序没有提供相应的支持，这对用户是不能接受的）。这类问题不一定都是错误，比如微软公司提供的Office宏的开发，是为了满足客户针对自身特色而设计的适合其专业工作的程序。

7）不能做用户期望的工作

例如，极少有人会期望一个本来编写用来对姓名进行排序的程序却按照ASCII码的顺

序排序，他们也不会指望用它来计算首位空格或区分大小写。当然用户名的排序还是要做的，问题是开发者需要重新构想一个新的排序规则来满足用户需求。

以下是在窗口中容易出现的功能性的错误：

(1) 遗漏“关于”功能，缺失对产品的介绍和版本信息等；

(2) 登录窗体应该包含公司的标志；

(3) MDI 程序应该能够恢复文档窗口的大小和位置；

(4) 不同界面中的同一功能应该使用同样的图标和图片。

2. 易用性

易用性主要针对 GUI 在使用时是否能清晰地表达产品的特征，用户方便地找到和使用系统中的各个功能。在 GB/T 16260—2006《软件工程产品质量》的第 1 部分中，提出易用性包含易理解性、易学习性和易操作性，即易用性是指在指定条件下使用时，软件产品被理解、学习、使用和吸引用户的能力。易用性测试包括针对应用程序的测试，同时还包括对用户手册系统文档的测试。通常采用质量外部模型来评价易用性。包括如下方面的测试：易理解性测试、易学性测试、易操作性测试、吸引性测试和易用的依从性测试。

下面是在窗口中容易出现的易用性方面的问题。

(1) 不同界面中的同一功能应该使用同样的图标和图片。

(2) 公司的系列产品要保持一致的界面风格，如背景色、字体、菜单排列方式、图标、安装过程、按钮和用语等应该大体一致。

(3) 子窗体的大小最好不要超过父窗体，且最好不要遮住父窗体的主要信息。如果存在多层嵌套窗口，每层窗口弹出时都自动往右下移动一点点，以保证不遮盖上层窗口标题为准。

(4) 窗口嵌套层次最好不超过 3 层。

(5) 单击窗口中的帮助按钮或按 F1 键必须带出和窗口内容相一致的帮助。

(6) 菜单和工具条应有清楚的界限。

(7) 菜单应采用“常用—主要—次要—工具—帮助”的位置排列。提供常用的菜单项，如“文件”“编辑”“查找”“打印”等。对常用的菜单项提供快捷命令方式。快捷方式唯一。

(8) 如果系统的模块较多、较深，经常会使用多级菜单，最好在窗口上加上导航条，以方便用户可以快速返回。

3. 美观性

美观性是指设计的图形界面具有良好的、美观的表现，能够提供更好的服务。

4. 用户满意度

用户满意度是对产品最终的衡量标准。一个令用户满意的产品，才是设计人员最终的目标。

5.6 GUI 测试指南

由于图形用户界面的普及，针对 GUI 的测试也单独成为了软件测试的一个重点。

在 GUI 刚开始被采用时，由于没有统一的规范，这一部分的测试比较主观。但随着 GUI 技术的成熟以及组件的大量采用和重用，越来越多可以遵循的指南使得 GUI 测试更加

客观,也更加贴近用户的使用习惯。

GUI 测试主要关注应用程序中的 GUI 组件是否符合规范或用户的操作习惯。当然,GUI 测试不能脱离功能测试而独立进行,它是随着功能测试的进展逐个窗口进行测试的。有时 GUI 测试也可以和功能测试一起进行,但对于稍微大一些的系统,最好将其分开,这样才不至于遗漏任何一个重点。

以下是作者收集、整理的一些通用的 GUI 测试方法,可供读者在实际操作中参考。

1. GUI 测试指南(标准、规范)

(1) 采用标准的键集(快捷键),在同一系统中,同样的操作,特别是名称相同的操作最好使用一致的快捷键。例如,浏览按钮如果在一个窗口中快捷键是 Alt+B,在另一个窗口最好采用同样的快捷键,这样可以方便用户的操作,不至于让用户混淆快捷键。除非在另一个窗口有比其更重要的操作已占用了一个快捷键,否则最好不要改变。

(2) 应用程序启动或进入系统的第一个界面应该显示“关于系统”或有关系统相关信息的屏幕。

(3) 应用程序应该保持为最大化。

(4) 应用程序可以在 Windows 的任务栏和状态栏中显示,如需要在系统托盘中显示的,在缩小至系统托盘和用户移动鼠标至应用程序的图标上时,最好给予相关的信息。

(5) 在系统中使用统一的代表应用程序的图标。

(6) 所有的窗口/对话框应具有可以和其他应用程序区分开的一致外观。

(7) 应用程序中使用的颜色组合应该有吸引力,且风格一致,搭配合理,色彩的跳跃不要太大。避免使用深色系,特别是红色和绿色,有些客户可能辨色困难。

(8) 登录界面上要有产品信息,如标志和版本,同时包含公司图标。

(9) 帮助菜单的“关于”中应有版权和产品信息。

(10) 应用程序中应按功能将界面划分为局域块,将完成相同或相近功能的按钮框起来,并配有相应的功能说明。

(11) 允许使用 Tab 键切换,且顺序与控件排列顺序要一致,目前流行总体从上到下,同时行间从左到右的方式;Tab 不能定位在不可见的控件上。

(12) 在不同的分辨率下显示正常(不出现水平和垂直滚动条,无截断的组件),特别是在应用程序推荐的分辨率下应显示完全正常,一般为 1024×768 和 800×600。

(13) 系统中所有的文字应该没有拼写/文字错误,句子没有语法错误,最好贴近用户的使用习惯。

(14) 使用用户知道的术语,而不是深奥的技术术语,特别是在错误提示的消息框中,让用户可以很快地知道问题的所在,而不是揣摩错误信息的意思。尽可能少用缩写。

(15) 英文系统中注意不要使用中文或其他语系字符。

(16) 在标识控件用途的标签文本和提示信息中,应使用半角符号。如果是指导性标签文本(如解释按钮功能的句子),则使用全角符号,并且句子应遵循中文标点符号使用规范。

(17) 在提示信息中,避免使用主语,尽可能使用被动语态。提示信息应简洁明了,没有侵犯性词语。使用一致的大小写规则,避免全大写和复杂的符号。

(18) 系统使用统一的字体,不要使用需要另外安装的或操作系统特定的字体库。注意斜体和粗体的使用。

(19) 系统目前不提供,以后版本才提供的功能最好隐藏,同一版本不同级别的系统中不允许使用的功能变灰,或操作提交时给予提示。

(20) 系统的帮助文件应该和当前的系统版本相一致。

(21) 使用“退出”命令终止程序;使用“关闭”移走主窗口和非模式对话框;使用“取消”移走模式对话框。当关闭主窗口并不表示终止进程时,对于主窗口使用“关闭”来代替使用“退出”。例如,关闭打印机状态窗口不会取消打印任务。

(22) 退出系统后应该彻底关闭程序,而不要在系统托盘或任务栏中继续保留系统的某个窗口。如果要保留,在退出系统时应该给予用户提示。

(23) 程序应该能够保存而恢复到用户最后退出的状态。MDI 程序应该能够恢复文档窗口的大小和位置。对话框和下拉框通常应该使用最后输入的值作为默认值。

(24) 不同界面中的同一功能应该使用同样的图标和图片。

(25) 公司的系列产品要保持一致的界面风格,如背景色、字体、菜单排列方式、图标、安装过程、按钮和用语等应该大体一致。

2. 按钮类 GUI 关注点

(1) 在同一窗口中实现某一功能的按钮是唯一的。

(2) OK 按钮总是在上方或者左方,而 Cancel 按钮总是在下方或右方。

(3) Cancel 按钮的等价按键通常是 Esc,而选中按钮的等价按键通常是 Enter。

(4) 测试按钮能否正常地实现功能,常用按钮的功能如下。OK:接受输入的数据或显示的响应信息,关掉窗口。Cancel:不接受输入的信息,关掉窗口。取消时最好给予提示,尤其在有大量输入的窗口时。Close:结束当前的任务,让程序继续进行,关掉数据窗口。Help:调出程序的帮助信息。Save:保存数据,停留在当前窗口。如果保存耗时长的话,最好显示类似沙漏或进度条之类的提示。注意验证能否重复保存,如在 IE 中由于网速慢而导致的重复保存。Add:新增记录。新增的记录必须排在首页首行。提交失败后必须保留用户已输入的内容,以便再次提交。提交时需对主要标识字段进行重复值、空值(空格)判断。Update/Edit:修改/编辑记录。如界面存在复选框,勾选多条记录进行修改时,需给予“只能对一条记录进行修改,默认为第一条”的提示信息。修改时加载的内容都为该记录的实际内容,而不再为默认值。修改完成后必须回到原记录所在位置,且刷新显示修改后的值。在查询条件下修改返回后如不满足查询条件则不显示。Delete:删除记录。在删除之前必须有确认删除的提示信息。删除成功后刷新不显示被删除的记录。删除成功后返回到原记录所在页面;而当原记录所在页不存在时,则返回上一页。当被删除的记录与其他记录存在关联时,应给予不允许删除及更详细的提示信息。针对大批量的删除应提供全选复选框,方便用户删除。Search:查询记录。每次查询应显示返回的结果数。每次查询应定位到首页。保留前一次的查询条件。当查询条件较多时,需配以重置按钮。当未查询到任何记录时,需给予未找到相关记录的提示信息。除用户明确要求不需要外,需提供模糊查询及组合查询功能。当查询返回的结果大于默认的一页大小时,最好采用分页或者根据系统默认或用户定义的一页显示的记录数量来分页。如有多页,需要提供首页、下一页、上一页、尾页和跳至指定页功能。每页的记录不能重复,但也可以根据用户需要显示上一页的最后一条数据。Reset:重置。应回到打开窗口时的最初状态。注意验证多次点击是否还能正常显示。Return:返回。如果一个窗口或页面不能通过菜单或工具栏到达,而是必须通过前一个窗

口完成才到达,应提供返回按钮或导航条让用户可以返回。

(5) 如果点击按钮后还需要用户的进一步的操作,按钮的名称应加上省略号。

(6) OK/Cancel/Apply/Help 按钮的排放最好遵从 Windows 的标准。

(7) 按钮最好都给予浮动提示,特别是图片按钮,这样可以避免由于网络太慢而导致的太长时间不能往下操作。

3. 信息处理类 GUI 测试关注点

(1) 窗口/屏幕上的每一个字段都应当有相应的标签。

(2) 根据文本框可以接受的类型测试文本框:①输入正常的字母或数字;②输入已存在的信息;③输入超过允许长度的字符或边界数字;④输入空白字符或空格;⑤输入不同类型或不同日期格式的数据;⑥用复制/粘贴等操作强制输入程序不允许的输入数据;⑦输入数据库或特殊字符集,例如 NULL 及/n 等。

(3) 测试文件选择框的正确性。使用空文件、只有空格的文件、不同类型的文件、同名文件、内容相同名称不同的文件以及大文件等。

(4) 测试强制性字段的正确性。强制性字段必须用红色的星号(*)标识,最好是必填项没有输入时,在光标移走时在相应的文本框后显示需要用户输入的红色信息。一般也可以在提交时用弹出消息框提示未填的必填项,关闭消息框后必须停留在第一个待输入的文本框中。

(5) 每一个新窗口/屏幕中,光标默认停留在第一个待输入的文本框中。

(6) 一般下拉框中应显示一个默认值,列表框中高亮度显示一个默认值。如果不需要默认值时,一般默认值为"请选择..."。

(7) 一般来说系统应记忆以前输入或选择的信息,但是当涉及安全时,最好不要保留用户的信息。有些地方可以使用复选框让用户决定是否要保留信息,如登录界面。

(8) 对输入信息类型有限制的文本框应在用户输入非法值后给予提示,对于日期型的输入框,最好在标签上就给予提示。

(9) 当输入的内容达到了字段的长度限制时,一般应控制不允许再输入,或者在提交后提示具体的允许输入的长度,而不是自动截断。

(10) 对于系统中不允许的非法字符,最好是在输入时不允许输入,或在提交时给予具体系统不允许的非法字符列表提示(如′、″、<、<>等)。

(11) 正确使用复选框或单选按钮。如果结果只有一个的,则使用单选按钮,如性别。验证单选按钮不能同时选中,只能选中一个,而可以选择多个复选框。

(12) 一组单选按钮在初始状态时必须有一个被默认选中,不能同时为空。

(13) 分别测试多个复选框可以被逐一选中、同时选中、部分选中以及都不被选中。

(14) 通过输入数字或用单击上下箭头来测试旋转按钮,测试其自动循环性,如范围为0~999 先输入 999,再单击向上箭头,看是否会跳到 0。输入字符或超过边界的数值,系统应该提示错误且重新输入。

(15) 验证列表框中的条目内容显示正确;允许多选的列表框,要分别检查按住 Shift 键和 Ctrl 键选中条目的情况。

(16) 避免使用水平滚动条,因为它会使项目阅读起来比较困难。解决的办法有:尽量使用垂直滚动条,加宽窗口,减小文本的宽度,或者使文本自动换行等。当然,如果确实需

要，还可以使用水平滚动条。

(17) 全选框选中时应该选中当前页所有记录，去掉当前页某个记录的勾选，则全选也不选中。翻页后，自动去掉已勾选的记录及全选的勾选。

(18) 复选框可以通过空格键控制选中/不选中。F4，Alt+Down 或 Alt+Up 键控制组合框打开和关闭。对于组合框，Escape 键等同于 Cancel 按钮，上、下箭头按钮控制向上或向下，Shift+Up 和 Shift+Down 键可以多选，Ctrl 键实现多选。

5.7 本章小结

本章介绍了性能测试、压力测试、容量测试、可靠性测试以及 GUI 测试。首先在性能测试中重点分析了反映系统性能的四个基本指标(响应时间、并发用户数、吞吐量、性能计数器)。其次介绍了压力和容量测试，这两个测试分别从不同的侧面反映了系统在极限负载情况下的工作能力。最后本章还引入了 GUI 测试的相关概念，以功能性、易用性、美观性和用户满意度等不同的角度介绍了 GUI 测试。最后给出了 GUI 测试指南，可供读者在实际测试过程中参考。

习　　题

1. 简述性能测试的 4 个主要基准。

2. 简述压力测试和容量测试的联系与区别。

3. 对一个长度为 100 000 条指令的程序进行测试，记录下来的数据如下：

测试开始，甲、乙两个测试人员同时测试指令。经过 160 小时的测试执行，甲测试人员累计发现 70 个错误，乙测试人员累计发现 80 个错误，甲乙二人共同发现 50 个错误，其中修复缺陷的时间为 10 个小时。又经过 160 小时的测试，甲测试人员累计发现 279 个错误，乙测试人员累计发现 290 个错误，甲乙二人共同发现 269 个错误，其中累计修复缺陷的时间为 20 个小时。

(1) 估计程序中固有的错误总数；

(2) 计算故障率和维修率；

(3) 计算平均无故障工作时间 MTBF、平均维修时间 MTTR 和有效度 A。

4. 什么是软件运行剖面？简述运行剖面对软件测试的影响。

第 6 章　测试流程与测试文档

本章主要介绍软件测试的流程以及在软件测试过程中需要提交的测试文档，通过本章的学习，读者应该对软件的测试文档有一个大致的了解，能够在软件测试的过程中提交正确、规范的测试文档。

6.1 测 试 流 程

软件生命周期分为需求阶段、概要设计阶段、详细设计阶段、编码阶段、测试阶段和运行/维护阶段。在这些不同阶段都有某种程度的重复，但各个阶段必须按一定的顺序结束，并提交相应的文档。

在各个阶段，首先要制定测试计划，然后确定测试任务，最后提交可交付文档，如表 6.1 所示。

表 6.1　软件生命周期测试流程

	测 试 输 入	测 试 任 务	可交付文档
需求阶段	软件质量保证计划 需求计划	制定验证和确认测试计划； 对需求进行分析和审查计划； 分析并设计基于需求的测试	验证测试计划； 针对需求的验证测试计划； 针对需求的验证测试报告
功能设计阶段	功能设计规格说明	功能设计验证和确认测试计划； 分析和审核功能设计规格说明； 可用性测试设计； 分析并设计基于功能的测试，构造相应的功能覆盖矩阵； 实施测试	(主确认)测试计划； 针对功能设计的验证测试计划； 针对功能设计的验证测试报告
详细设计阶段	详细设计规格说明	详细设计验证测试计划； 分析和审核详细设计规格说明； 分析并设计基于内部的测试	详细确认测试计划； 针对详细设计的验证测试计划； 针对详细设计的验证测试报告
编码阶段	代码清单	代码验证测试计划； 分析代码； 验证代码； 设计基于外部的测试； 设计基于内部的测试	测试用例规格说明； 需求覆盖或跟踪矩阵； 功能覆盖矩阵； 针对代码验证测试计划； 针对代码验证测试报告
测试阶段	要测试的软件用户手册	制定测试计划； 审查开发部门进行的单元和集成测试； 进行功能测试； 进行系统测试； 审查用户手册	测试记录； 测试事故报告； 测试总结报告
运 行/维护阶段	已确认的问题报告 软件生存周期过程	监视验收测试； 为确认的问题开发新的测试用例； 对测试进行有效性评估。	可升级的测试用例库

具体的测试流程可以分为以下步骤：软件需求分析→设计测试计划(获得测试需求→确定测试策略)→设计测试用例→搭建测试环境→执行测试用例→bug统计分析→出具测试分析报告。

1. 需求分析

软件测试中的一个重要环节就是需求分析，测试开发人员首先要对这个环节正确理解，才能进行接下来的测试工作。需求分析是至关重要的。

一般而言，需求分析包括软件功能需求分析、测试环境需求分析和测试资源需求分析等。

其中最基本的是软件功能需求分析，测试一个软件，首先要知道该软件能实现哪些功能以及是怎样实现的。比如，图书馆管理系统包括借书、还书等功能，那么就应该知道软件是怎样实现这些功能的，为了实现这些功能需要哪些测试设备以及如何搭建相应测试环境等，否则测试就无从谈起。

那么根据什么来做测试需求分析呢？总的来说，做测试需求分析的依据有软件需求文档、软件规格书以及开发人员的设计文档等，管理规范的公司在软件开发过程中都有这些文档。

2. 测试计划

软件测试计划需要定义对什么进行测试、测试哪些方面、如何执行测试，以及谁来执行测试。因此，测试计划在更大程度上是一个管理性文档，其目标读者是掌握适当技术知识的管理者。制定测试计划要依据软件实现的相关细节，例如编程语言、逻辑和类型等。理想情况下，测试计划应该包含静态测试和动态测试。

测试计划一般由测试负责人编写。制定测试计划的依据主要是项目开发计划和测试需求分析结果。测试计划一般包括以下一些方面。

(1) 测试背景。

① 软件项目介绍；

② 项目涉及人员(如软硬件项目负责人等)介绍以及相应联系方式等。

(2) 测试依据。

① 软件需求文档；

② 软件规格书；

③ 软件设计文档；

④ 其他(如参考产品等)。

(3) 测试资源。

① 测试设备需求；

② 测试人员需求；

③ 测试环境需求；

④ 其他。

(4) 测试策略。

① 采用的测试方法；

② 搭建哪些测试环境；

③ 采取哪些测试工具和测试管理工具；

④ 对测试人员进行培训等。

(5) 测试日程。

① 测试需求分析；

② 测试用例编写；

③ 测试实施，根据项目计划，测试分成哪些测试阶段(如单元测试、集成测试、系统测试阶段、α/β测试阶段等)，每个阶段的工作重点以及投入资源等。

(6) 其他。

测试计划还要包括编写日期、作者等信息，计划要尽可能地详细。

一份计划做得再好，当实际实施的时候就会发现往往很难按照原有计划开展。如在软件开发过程中资源匮乏、人员流动等都会对测试造成一定的影响。所以，这些就要求测试负责人能够从宏观上加以调控，在变化面前能够做到应对自如、处变不惊。

3. 测试设计

测试设计主要包括测试用例编写和测试场景设计两方面。一份好的测试用例对测试有很好的指导作用，能够发现很多软件问题。在测试工作开展前完成测试用例的编写，可以避免测试工作的盲目性，测试用例的存在可以大大降低测试的工作量，从而提高测试的工作效率。测试场景设计则是测试环境的设计。

4. 测试环境搭建

不同软件产品对测试环境有着不同的要求。例如对于一些嵌入式软件，如手机软件，如果想测试有关功能模块的耗电情况、手机待机时间等，可能就需要搭建相应的电流测试环境了。当然测试中对于如手机网络等环境都有所要求。

测试环境很重要，符合要求的测试环境能够帮助测试人员准确地测出软件的问题，并且做出正确的判断。

5. 测试执行

测试执行的目的是发现软件中存在的缺陷，由测试人员来执行，开始执行前，必须考虑测试执行前相关工作是否已经准备就绪。执行的时候根据《测试需求文档》、《测试计划》、《测试执行计划》和《测试用例》进行。执行分为以下阶段：单元测试→集成测试→系统测试→出厂测试，其中每个阶段还有回归测试等。

6. 测试记录

缺陷记录是软件测试生命周期中最重要的可用产出之一。一条好的缺陷记录能够减少测试人员和开发人员的沟通成本，加快缺陷修复的速度，增强测试的可信度。

缺陷记录的最终目的是准确地传达测试人员的思想或缺陷的真正所在。只要遵循本规范中的一些简单原则，就可以轻松地填好每一条缺陷记录，从而提高工作效率。缺陷记录总的说来包括两方面：由谁提交和缺陷描述。一般而言，缺陷都是谁测试谁提交，当然有些公司为了保证所提交缺陷的质量，还会在提交前进行缺陷评估，以确保所提交的缺陷的准确性。

7. 软件评估

软件经过一轮又一轮的测试后，在确认软件无重大问题或者问题很少的情况下，对准备发给客户的软件进行评估，以确定是否能够发行给客户或投放市场。

软件评估小组一般由项目负责人、营销人员和部门经理等组成，也可能是由客户指定的第三方人员组成。

8. 测试总结

每个阶段和每个版本都要有相应的测试总结，当项目完成后，一般要对整个项目做回顾总结，看有哪些做的不足的地方，有哪些经验可以供今后的测试工作借鉴使用等。

9. 测试维护

由于测试的不完全性，当软件正式投入使用后，客户在使用过程中难免遇到一些问题，有的甚至是严重性的问题，这就需要修改有关问题，修改后需要再次对软件进行测试、评估和发行。

6.2 测试文档的编写

项目测试文档是用来记录、描述、展示测试过程中一系列测试信息的处理过程，通过书面或图示的形式对项目测试活动过程或结果进行描述、定义及报告。例如，分阶段测试计划文档、测试流程文档、测试数据文档、测试参数设置文档和测试指南文档等。这些文档将会伴随着软件测试的各个阶段逐渐充实、完善，同时也记载了整个测试的过程和成果。

作为测试技术人员，在测试过程中应将各种标准测试文档提交给项目组，以确保软件测试项目的质量。也就是说，测试技术人员的工作绩效与文档的高质量提交息息相关，它描述项目测试过程的每一个细节。因此，从某种程度上讲，测试管理的核心其实就是测试文档管理。

(1) 测试文档有助于项目测试水平的提高。

从内容上说，项目测试文档大致可以分为测试成果文档和测试过程文档两大类。测试成果文档作为项目可交付物的一个组成部分，其重要性自然不言而喻。测试过程文档主要记录了项目测试过程中的各种信息，为测试人员提供决策依据，以保证项目的顺利实现。另一方面，测试过程文档也是测试过程最为宝贵的资产，通过对测试过程文档进行归纳和分析，可以对测试项目的成功经验和失败教训了然于胸，从而使后续的测试运作更加有的放矢。

(2) 测试文档驱动着测试过程。

测试项目的阶段性成果是以测试文档形式体现的，因此，测试项目的运作在一定程度上是由测试文档驱动的。从测试文档的角度来看，项目测试过程就是一个文档制作与执行的过程。在项目测试的过程中，每项工作的事前计划、事中测试记录和事后分析结果都要形成相应的测试文档，文档包括与项目相关的资源及其使用情况。

因此，测试文档是软件项目的一部分，没有正式的测试文档的活动，就不是规范的测试。测试文档的编制和管理在项目测试中占有突出的地位和相当大的工作量，高质量地编制、变更、修正、管理和维护文档，对于提高项目测试的质量和客户满意度有着重要的现实意义。

测试文档是否专业已成为测试管理和测试人员的重要评价指标之一。测试文档普遍存在以下缺点：

（1）文档编写不够规范。主要是测试文档内容描写不够完善，在编写各种测试文档的过程中，虽然大家都按事先规定的模式进行了编写，但编写的内容不够完善。要么文档极其简单要么文档流于形式，没有什么实际的价值；甚至有的测试文档与测试过程完全不符。

（2）测试文档没有统一入库管理。随着软件开发的不断深入、升级，新错误不断产生，各种测试文档越来越多，但没有建立一个测试文档资料库。在测试过程中没有对每一个阶段的文档进行整理和分层次管理，各类文档资料缺少一致性。不同时期的各种测试文档零散存在，造成查询测试文档时非常困难。在众多的测试文档中，其中一些文档必定是关键文档，起到非常重要的作用，对于这类测试文档没有设定优先级特别说明。

（3）只重视测试文档的形式，实用性不强。在实际的测试过程中，编制人员没有时间去关心它们的用途，也不知道哪些部门使用它们，更注重的是在规定的时间内完成任务，以免影响考核成绩。这样一来，一些不实用的、重复的文档不但阻碍着测试的执行效率，而且影响项目的整体进度。因此，文档的制定要实用，以减少繁文缛节的文字工作。

在实际的测试文档的编写中，必须克服以上缺点，编写出高效的、准确的，真正能够驱动测试项目运作的文档。

软件测试结束以后，最终的工件一般有3个：测试计划、测试用例和测试结果报告。测试计划中包括了测试的背景、人员和内容，以及计划要做的测试；测试用例是对于计划中要做的测试内容和测试项生成的用例；测试结果包括了用例的测试结果和总结，以便将来维护时使用。整个测试过程中这3个工件都应该是不断被更新的，只有一个最终版本。

6.2.1 测试计划编写

软件测试是一个有组织、有计划的活动，应该给予充分的时间和资源进行测试计划，这样软件测试才能在合理的控制下正常进行。测试计划是软件测试中最重要的步骤之一，它在软件开发的前期对软件测试做出清晰、完整的计划，不仅对整个测试起到关键性的作用，而且对开发人员的开发工作、整个项目的规划和项目经理的审查都有辅助性作用。

测试计划文档可以确定现有项目的信息和应测试的软件构件；列出推荐的测试需求；推荐可采用的测试策略，并对这些策略加以说明；确定所需的资源，并对测试的工作量进行估计；列出测试项目的可交付元素。

测试计划阶段的测试文档是指明测试范围、方法、资源以及相应测试活动的时间进度安排表的文档。测试计划文档应该包含以下部分：

（1）目标。表示该测试计划应达到的目标。

（2）在概述部分中明确项目背景和范围，简要描述项目背景以及所要求达到的目标，如项目的主要功能特征、体系结构及简要历史等，指明该计划的适用对象及范围。

（3）测试对象。列出所有将被作为测试目标的测试项，包括功能需求、非功能需求、性能及可移植性等。

测试计划文档应该有一个封面。包括名称、版本号、编写人、审批人和审批日期。测试计划文档大纲要包含以下内容。

1 简　　介

1.1 编写目的

对测试计划做简单的介绍，说明这个测试计划的功效以及当前项目背景情况介绍。对测试产品(所属行业、系统架构和系统功能等)及其项目目标、该文档读者对象以及其他相关事项进行一个简单说明。

1.2 名词解释

项目中或测试中一些术语的说明，包括使用的专用术语及其定义和缩略语全称及其定义。

1.3 测试摘要

主要说明测试计划中重要的和可能有争议的问题。主要目的是将这些信息传递给那些可能不会通读整个测试计划文档的人员(比如公司领导、项目经理和产品经理等)，包括重点事项、争议事项、风险评估和时间进度等。

1.4 参考资料

包括测试计划引用或参考的文档，查看计划同时需要查看的相关文档等，这些文档都需要加入测试计划的参考资料列表中。

1.5 测试范围

本计划涵盖的测试范围，比如功能测试、集成测试、性能测试和安全测试等，测试项目涉及的业务功能与其他项目涉及的业务接口等。要说明哪些是要测试的，哪些是不要测试的；哪些文档需要编写，哪些文档在什么情况下可以不写等。

2 测试说明

2.1 测试项说明

2.1.1 系统名称

2.1.2 应测试项

2.1.3 非测试项

2.2 测试资源

2.2.1 硬件设备

描述建立测试环境所需要的设备、用途及软件部署计划，例如配置、用途及特殊说明、软件及版本和预计空间等。

2.2.2 软件设备

列出项目中使用的所有软件及测试工具。

2.2.3 人力资源

列出项目参与人员的职务、姓名和职责。人员包括开发人员、配置、测试以及其他相关人员。

2.2.4 测试工具

2.3 测试安排

2.3.1 测试培训

2.3.2 测试进度

对各阶段的测试给出里程碑计划，包括阶段、里程碑、资源等。

一般测试进度用表格的形式详细制定，如表6.2所示。

表6.2 测试进度表

测试阶段	开始时间	结束时间	资源	是否里程碑
系统测试计划				
测试用例编写				
测试用例评审				
单元测试				
用户手册编写				
集成测试				
系统测试				
系统测试报告编写				

2.4 测试文档

3 风险和约束

列出测试过程中可能存在的一些风险和制约因素，并给出规避方案。例如，由于客观存在的设备、网络等资源原因，使得测试不全面，明确说明哪些资源欠缺，产生什么约束；由于研发模式为项目型产品，且工程上线时间压力大，使得测试不充分，明确说明在此约束下，测试如何应对；由于开发人员兼职其他工作，造成所提交的代码质量以及不能及时修改错误的风险，明确说明测试应该如何应对。

4 测试优先级

为了测试计划的目的，在项目版本的进度下，测试执行的组织和安排测试用例将帮助测试人员达到这些目标。作为这种组织的一部分，要考虑每一个测试用例的优先级别，根据优先级别对测试用例进行分组可以帮助测试人员决定不同类型的版本需要什么样的测试用例，因此应该计算需要的时间。如果只有有限的时间，可以查看哪种方案是最合适的。

5 测试策略

5.1 整体策略

说明计划中使用的基本的测试过程。使用里程碑技术在测试过程中验证每个模块，测试人员在需求阶段参与测试工作，进行需求回顾、设计回顾、测试用例设计和测试开发，在系统开发完成之后，正式执行测试。产品达到软件产品质量要求和测试要求后发布，并提交相关的测试文档。

5.2 测试类型

选择本项目是否采用各个测试类型。如果表格中没有对应的测试类型，可以自己增加测试类型，如表6.3所示。

表 6.3 测试类型表

编号	测试类型	说　明	是否采用
1	接口测试	检查系统与外部系统或外部设备等的接口是否正常	
2	集成测试	对应用系统的各个部件进行联合测试	
3	数据和数据库完整性测试	以数据库表为单位,检查数据库表以及表中各字段命名是否符合命名规范,表中字段是否完整,字段描述是否正确,包括字段的类型、长度、是否为空,数据库表中的关系、索引、主键和约束是否正确	
4	功能测试	根据需求文档、设计文档等检查产品是否正确实现了功能	
5	流程测试	按操作流程进行的测试,主要有业务流程、数据流程、逻辑流程和正反流程,检查软件在按流程操作时是否能够正确处理	
6	用户界面测试	检查界面是否符合公司界面规范,是否美观合理	
7	性能测试	提取系统性能数据,检查系统是否满足在需求中所规定应达到的性能	
8	安全性测试	检查系统安全是否达到安全需求,是否存在安全隐患	
9	故障转移和恢复测试	确保测试对象能成功完成故障转移,并从硬件、软件或网络等方面的各种故障中恢复,这些故障导致数据意外丢失或破坏了数据的完整性	
10	配置测试	测试目标软件在具体硬件配置情况下是否出现问题	
11	易用性测试	检查系统是否易用友好,是否符合通用的操作习惯	
12	安装测试	检查系统能否正确安装,配置基础数据是否正确	
13	兼容性测试	对于C/S架构的系统,需要考虑客户端支持的系统平台;对于B/S架构的系统,需要考虑用户端浏览器的版本	
14			
15			

5.3 测试技术

选择本项目是否采用各种测试技术。如果表格中没有对应的测试技术,可以自己增加测试技术,如表6.4所示。

表 6.4 测试技术表

编号	测试技术	说　明	是否采用
1	测试用例设计	在产品需求评审通过后编写测试用例	
2	白盒测试	单元测试是否开展代码测试	
3	自动化测试	系统回归时是否要引入自动化测试	
4	性能测试	是否使用工具进行性能方面的测试	

6 测试提交文档

测试过程中需要提交各种文档、作者以及文档配置库存放目录，如表6.5所示。

表6.5 测试提交文档表

文档说明	作者	文档位置(配置库)
系统测试计划		
测试用例		
实现与测试跟踪表		
用户手册		
系统测试报告		

7 质量目标

说明可以使产品的质量达到什么样的目标，产品的流程联通性达到什么样的要求，如表6.6所示。

表6.6 测量质量目标表

编号	测试质量目标	确认人以及特殊说明
1	测试已实现的产品是否达到设计的要求，包括：各个功能点是否已实现，业务流程是否正确	
2	所有的测试用例已经执行通过	
3	所有的自动测试脚本已经执行通过	
4	不允许存在严重程度为高和中的功能缺陷	
5	缺陷的发现速率正在下降并接近0	
6	在最后的三天内没有发现严重程度为高和中的缺陷	
⋮		

8 计划审核记录

包括质管部经理和项目经理等的审核意见。

这是一个很全面的测试计划文档的模板，适合于大的测试项目。如果只是做功能的测试，或者只是做一个数据迁移的测试，涉及大的测试项没有这么多，可以根据具体测试工作任务情况编写测试计划，剪裁适合自己的测试计划模板。

表6.7是比较小的测试任务使用的测试计划模板。

测试计划书并不是一成不变的，随着项目的进行，会由于各方面的因素，例如，提交测试的程序版本质量低，错误量大，修改慢，需求发生变更等等，导致测试计划无法按原计划执行，此时就应该适当的调整测试计划。从文档的角度来看，测试计划书是最重要的测试文档，完整细致并具有远见性的测试计划书会使测试活动安全顺利地向前进行，从而确保所开发的软件产品的高质量。

表 6.7 测试计划模板

测 试 计 划			
测试计划概述			
项目名称		测试开始日期	
测试产品版本		提交版本日期	
开发人员		测试人员	
产品经理		发布日期	
测试阶段	测试周期		任务安排
第一阶段			
第二阶段			
第三阶段			
第四阶段			
第五阶段			
第六阶段			
参考文档			
测试环境			
测试需求描述	测试项描述	期望结果	优先级别

6.2.2 测试用例编写

软件测试用例就是指导测试人员对软件执行操作，帮助测试人员证明软件功能或发现软件缺陷的一种说明，是软件测试的核心。对于测试人员来说，测试用例的编写是一项必须掌握的能力。但有效的设计和熟练的编写却是一个十分复杂的技术，它需要测试人员对整个软件不管从业务还是从功能上都有明晰的把握，测试用例必须根据被测项目的真实情况来编写才能起到真正的作用。下面介绍测试用例的编写内容。

1. 引言

编写目的：通过测试尽可能找出项目中的错误，并加以纠正。测试不仅是最后的复审，更是保证软件质量的关键。

项目背景：包括系统说明、项目开发小组、主管科目和任务下达者等一系列和项目相关

的背景内容。

参考资料：编写测试用例时需要参考的文档资料，例如项目的计划任务书、项目开发计划、需求规格说明书、概要设计说明书、测试计划和用户操作手册等。

测试种类的分类：功能测试、健壮性测试、接口测试、强度测试、压力测试、性能测试、用户界面测试、安全测试、可靠性测试、安装/反安装测试和文档测试等。

测试阶段：包括功能测试、路径测试和界面测试等。

测试用例的种类：功能测试用例、路径测试用例和界面测试用例。

用例编写方案：如表6.8所示。

表6.8 用例编写方案

开发阶段	依据文档	编写的用例
需求分析结束后	需求文档	系统测试对应的用例
概要设计阶段结束后	概要设计、体系设计	集成测试对应的用例
详细设计阶段	详细设计文档	单元测试对应的用例

2. 测试用例

1）一般测试用例的编写方法

一般测试用例的编写模板如表6.9所示。

表6.9 测试用例表

<table>
<tr><td>用例编号</td><td colspan="2"></td><td colspan="2">测试优先级</td><td></td></tr>
<tr><td>用例摘要</td><td colspan="5"></td></tr>
<tr><td>测试类型</td><td colspan="5"></td></tr>
<tr><td>用例类型</td><td colspan="5"></td></tr>
<tr><td>用例设计者</td><td></td><td>设计日期</td><td></td><td>对应需求编号</td><td></td></tr>
<tr><td>对应 UI</td><td colspan="5"></td></tr>
<tr><td>对应 UC</td><td colspan="5"></td></tr>
<tr><td>版本号</td><td colspan="2"></td><td colspan="2">对应开发人员</td><td></td></tr>
<tr><td>前置条件</td><td colspan="5"></td></tr>
<tr><td>测试方法</td><td colspan="5"></td></tr>
<tr><td>输入数据</td><td colspan="5"></td></tr>
<tr><td>执行步骤</td><td colspan="5"></td></tr>
<tr><td>预期输出</td><td colspan="5"></td></tr>
<tr><td>实际结果</td><td colspan="5"></td></tr>
<tr><td>测试日期</td><td colspan="5"></td></tr>
<tr><td>结论</td><td colspan="5"></td></tr>
</table>

2）功能测试用例

功能测试用例用于对系统中各个模块的功能进行测试，来检测系统功能的健壮性和完

整性等。一般的模板如表 6.10 所示。

表 6.10　功能测试用例模板

模块名						
开发人员		版本号				
用例作者		设计日期				
测试类型	功能测试	测试工具				
用例 ID	用例名称	测试目的	输入描述	预期结果	实际结果	测试数据

3）接口-路径测试用例

接口测试用例模板如表 6.11 所示。

表 6.11　接口测试用例模板

接口 A 的函数原型			
输入/动作		期望的输出/响应	实际情况
典型值			
边界值			
异常值			
接口 B 的函数原型			
输入/动作		期望的输出/响应	实际情况
典型值			
边界值			
异常值			
⋮			

路径测试用例模板如表 6.12 所示。

表 6.12　路径测试用例模板

检　查　项		结论
数据类型问题	变量的数据类型有错误吗	
	存在不同数据类型的赋值吗	
	存在不同数据类型的比较吗	
变量值问题	变量的初始化或默认值有错误吗	
	变量发生上溢或下溢吗	
	变量的精度不够吗	

续表

检　查　项		结论
逻辑判断问题	由于精度问题导致比较无效吗	
	表达式中的优先级有误吗	
	逻辑判断结果颠倒吗	
循环问题	循环终止条件不正确吗	
	无法正常终止(死循环)吗	
	错误地修改循环变量吗	
	存在误差累积吗	
内存问题	内存没有被正确地初始化却被使用吗	
	内存被释放后却继续被使用吗	
	内存泄露吗	
	内存越界吗	
	出现野指针吗	
文件 I/O 问题	对不存在的或者错误的文件进行操作吗	
	文件以不正确的方式打开吗	
	文件结束判断不正确吗	
	没有正确地关闭文件吗	
错误处理问题	忘记进行错误处理吗	
	错误处理程序块一直没有机会被运行	
	错误处理程序块本身就有毛病吗(如报告的错误与实际错误不一致,处理方式不正确等等)	
	错误处理程序块是“马后炮”吗(如在被它调用之前软件已经出错)	

4）性能测试用例

性能测试用例主要是对被测试对象(单元)的介绍,说明测试的范围和目的,对测试环境和测试辅助工具进行描述,并进行测试驱动程序的设计。性能测试用例模板如表 6.13 所示。

表 6.13　性能测试用例模板

测试用例	
测试用例 ID	
性能描述	
用例目的	
前提条件	
特殊的规程说明	

续表

用例间的依赖关系				
步骤	**输入/动作**	**期望的性能(平均值)**	**实际性能(平均值)**	**回归测试**
1.	示例：典型值…			
2.	示例：边界值…			
3.	示例：异常值…			
4.	⋮			

5）图形用户界面测试用例

界面是软件与用户交互的最直接一层，界面的好坏决定用户对软件的第一印象。设计良好的界面能够引导用户自己完成相应的操作，起到向导的作用。测试人员必须设计良好的测试用例，以测试软件是否能够让用户进行轻松舒适的浏览以及进行更深入的操作。图形用户界面测试用例模板如表 6.14 所示。

表 6.14　图形用户界面测试用例模板

<table>
<tr><td>模块名称</td><td colspan="2"></td><td>版本号</td><td></td><td>日期</td><td></td></tr>
<tr><td>开发人员</td><td colspan="2"></td><td>设计人</td><td></td><td>测试人</td><td></td></tr>
<tr><td>功能描述</td><td colspan="2"></td><td colspan="2"></td><td colspan="2"></td></tr>
<tr><td>用例目的</td><td colspan="2"></td><td colspan="2"></td><td colspan="2"></td></tr>
<tr><td>前提条件</td><td colspan="2"></td><td colspan="2"></td><td colspan="2"></td></tr>
<tr><td rowspan="2">用例编号</td><td colspan="2">用例实施</td><td colspan="2" rowspan="2">期望输出</td><td colspan="2" rowspan="2">实际情况</td></tr>
<tr><td>动作</td><td>输入</td></tr>
<tr><td rowspan="6">易用性测试</td><td colspan="2">是否有快捷键、热键，并且已经设定的快捷键和热键不能重复</td><td colspan="2" rowspan="6"></td><td colspan="2" rowspan="6"></td></tr>
<tr><td colspan="2">减少用户输入动作数据量</td></tr>
<tr><td colspan="2">同一界面的控件数最好不要超过 10 个</td></tr>
<tr><td colspan="2">消除冗余输入，最好在程序中自动获取</td></tr>
<tr><td colspan="2">完成同一功能或任务的元素放在集中位置</td></tr>
<tr><td colspan="2">⋮</td></tr>
<tr><td rowspan="5">窗口测试</td><td colspan="2">窗口能否正常关闭</td><td colspan="2" rowspan="5"></td><td colspan="2" rowspan="5"></td></tr>
<tr><td colspan="2">窗口控件的布局</td></tr>
<tr><td colspan="2">活动窗口反显加亮</td></tr>
<tr><td colspan="2">多窗口重叠时窗口名称显示正确</td></tr>
<tr><td colspan="2">⋮</td></tr>
</table>

续表

用例编号	用例实施		期望输出	实际情况
	动作	输入		
菜单测试	菜单功能是否正确执行			
	下拉菜单是否根据选项含义进行分组			
	菜单命令快捷方式			
	菜单功能名称是否具有自解释性			
	⋮			
图标测试	图表基调颜色不刺眼			
	用户登录界面容易找到			
	图标符合常规表达习惯			
	图标上是否加标注			
	⋮			
鼠标测试	多次点击能否在语境中正确识别			
	光标、识别指针随操作恰当改变			
	支持滑轮			
	相同种类的元素采用相同的操作激活			
	⋮			
文字测试	文字易认易懂，拼写正确，不存在二义性			
	全角、半角明确，中英文不能混合			
帮助系统测试	系统提供 F1 帮助键			
	⋮			

6）健壮性测试用例

健壮性测试用例主要对被测试对象进行介绍，对测试环境和测试辅助工具进行描述，给出容错能力/恢复能力测试用例表。健壮性测试用例模板如表 6.15 所示。

表 6.15　健壮性测试用例模板

异常输入/动作	容错能力/恢复能力	造成的危害、损失
示例：错误的数据类型		
示例：定义域外的值		
示例：错误的操作顺序		
示例：异常中断通信		
示例：异常关闭某个功能		
示例：负荷超出了极限		
⋮		

7）可靠性测试用例

可靠性测试用例模板如表 6.16 所示。

表 6.16 可靠性测试用例模板

任务 A 描述	
连续运行时间	
故障发生的时刻	故障描述
……	
任务 A 统计分析	
任务 A 无故障运行的平均时间间隔	（CPU 小时）
任务 A 无故障运行的最小时间间隔	（CPU 小时）
任务 A 无故障运行的最大时间间隔	（CPU 小时）
任务 B 描述	

8）安装/反安装测试用例

安装/反安装测试用例模板如表 6.17 所示。

表 6.17 安装/反安装测试用例模板

配置说明		
安装选项	描述是否正常	使用难易程度
全部		
部分		
升级		
其他		
反安装选项	描述是否正常	使用难易程度

9）测试用例清单

测试用例清单模板如表 6.18 所示。

表 6.18 测试用例清单

项目 ID	测试项目	子项目 ID	测试子项目	测试用例 ID	测试结果	结论
总数						

3. 附录

测试组长邀请开发人员和同行专家，对系统测试用例进行技术评审，并给出评审意见，如表6.19所示。

表6.19 评审意见表

序号	评审人员	评审意见
评审总结	总结人： 时间：	

6.2.3 测试报告编写

测试报告是测试阶段最后的文档产出物，优秀的测试经理应该具备良好的文档编写能力。测试报告就是把测试的过程和结果写成文档，并对发现的问题和缺陷进行分析，为纠正软件存在的质量问题提供依据，同时为软件验收和交付打下基础。一份详细的测试报告应包含足够的信息，包括产品质量和测试过程的评价，测试报告基于测试中的数据采集以及对最终的测试结果的分析。

下面给出一个测试报告的模板，根据实际情况可以有所调整。

1 简 介

1.1 编写目的

本测试报告的具体编写目的，指出预期的读者范围。

1.2 项目背景

对项目目标和目的进行简要说明。必要时包括简史。这部分不需要脑力劳动，直接从需求或者招标文件中复制即可。

1.3 系统简介

如果设计说明书有此部分，复制即可。注意，必要的框架图和网络拓扑图能吸引眼球。

1.4 术语和缩写词

列出设计本系统项目的专用术语和缩略语约定。对于技术相关的名词和多义词一定要注释清楚，以便阅读时不会产生歧义。

1.5 参考资料

需求、设计、测试用例、手册以及其他项目文档都是可参考内容，另外还有测试使用的国家标准、行业指标、公司规范和质量手册等。

2 测试概要

测试的概要介绍，包括测试的一些声明、测试范围和测试目的等，主要是测试情况简介（其他测试经理和质量人员关注的部分）。

2.1 测试用例设计

简要介绍测试用例的设计方法，例如等价类划分、边界值和因果图等。

提示：如果能够对设计进行具体说明，在其他开发人员和测试经理阅读的时候就容易对用例设计有一个整体的概念。这样在没有看到测试结论之前，就可以了解到测试用例的设计技术。

2.2 测试环境与配置

简要介绍测试环境及其配置。如果系统/项目比较大，则用表格方式列出，如表6.20所示。

表6.20 测试环境/配置表

数据库服务器配置		客户端配置	
内存		CPU	
操作系统		硬盘	
机器网络名		应用软件	
应用服务器配置		局域网地址	
⋮		⋮	

对于网络设备和要求也可以使用类似的表格。对于三层架构的网络，可以根据网络拓扑图列出相关配置。

2.3 测试方法(工具)

简要介绍测试中采用的方法和工具。

提示：主要是黑盒测试，测试方法可以写上测试的重点和采用的测试模式，这样可以一目了然地知道是否遗漏了重要的测试点和关键模块。工具为可选项，当使用到测试工具和相关工具时要说明。注意，要注明是自产还是来自厂商，版本号多少，在测试报告发布后要避免工具的版权问题。

3 测试结果及缺陷分析

这是整个测试报告中最激动人心的部分，这部分主要汇总各种数据并进行度量，度量包括对测试过程的度量和能力评估、对软件产品的质量度量和产品评估。对于不需要过程度量或者相对较小的项目，例如用于验收时提交给用户的测试报告以及小型项目的测试报告，可省略过程方面的度量部分；而采用了CMM/ISO或者其他工程标准过程，需要提供过程改进建议和参考的测试报告(主要用于公司内部测试改进和缺陷预防机制)，则需要列出过程度量。

3.1 测试执行情况与记录

描述测试资源消耗情况，记录实际数据(测试经理和项目经理关注的部分)。

3.1.1 测试组织

可列出简单的测试组织架构图，包括测试组织架构、测试经理、主要测试人员和参与测试人员等。

3.1.2 测试时间

列出测试的跨度和工作量，最好区分测试文档和活动的时间。数据可供过程度量使用。

3.1.3　测试版本

给出测试的版本，如果是最终报告，可能要报告测试次数和回归测试的次数。

3.2　覆盖分析

覆盖分析包括需求覆盖和测试覆盖等。

3.2.1　需求覆盖

需求覆盖是指经过测试的需求，如表6.21所示。

表6.21　需求覆盖表

需求/功能(或编号)	测试类型	是否通过	需求覆盖率/%	备注

需求覆盖率是指经过测试的需求/功能和需求规格说明书中所有需求/功能的比值，通常情况下要达到100%的目标。需求覆盖率的计算方法是：经过测试的需求数/需求总数。根据测试结果，按编号给出每一测试需求是否通过的结论。

3.2.2　测试覆盖

测试覆盖率的计算方法是：执行的用例数/用例总数，如表6.22所示。

表6.22　测试覆盖表

需求/功能(编号)	用例个数	执行总数	未执行	未/漏测分析和原因	测试覆盖率/%

3.3　缺陷的统计与分析

缺陷统计主要涉及被测系统的质量，因此，这部分成为开发人员和质量人员重点关注的部分。所有缺陷的汇总除了可以用表格表示以外，还可以给出缺陷的饼状图或柱状图以便直观查看。

3.3.1　缺陷汇总

缺陷按严重程度汇总，如表6.23所示。

表6.23　缺陷汇总表

被测系统缺陷严重程度	严重	一般	微小
缺陷数量			

缺陷按类型汇总如表6.24所示。

表6.24　缺陷类型汇总表

被测系统缺陷类型	用户界面	一致性	功能	算法	接口	文档	其他
缺陷数量							

缺陷还可以按照功能分布汇总，如表6.25所示。

表6.25　缺陷功能汇总表

被测系统缺陷所属的功能	功能1	功能2	功能3	功能4	功能5	功能6	功能7	…
缺陷数量								

3.3.2　缺陷分析

本部分对上述缺陷和其他收集数据进行综合分析。

3.3.3　残留缺陷与未解决问题

本部分主要包括以下内容：

(1) 缺陷概要：该缺陷描述的事实。

(2) 原因分析：分析如何引起缺陷以及缺陷的后果，描述造成软件局限性和其他限制性的原因、预防和改进措施、弥补手段和长期策略。

(3) 未解决问题。

4　测试结论与建议

本部分是总结，对上述过程和缺陷进行分析之后给出结论，本部分为项目经理、部门经理以及高层经理关注的内容，需要清晰扼要地作结论。

4.1　测试结论

测试执行是否充分(可以增加对安全性、可靠性、可维护性和功能性的描述)，对测试风险的控制措施和成效、测试目标是否完成、测试是否通过以及是否可以进入下一阶段项目目标作出结论。

4.2　建议

对系统存在问题的说明，描述测试所揭露的软件缺陷和不足，以及可能给软件实施和运行带来的影响；指出可能存在的潜在缺陷和后续工作；对缺陷修改和产品设计提出建议；对过程改进方面提出建议。

测试报告的内容大同小异。对于一些测试报告而言，可能将第3部分和第4部分合并，逐项列出测试项、缺陷、分析和建议，这种写法也比较多见，尤其在第三方评测报告中。上面的报告模板仅供参考。

6.3　本章小结

本章首先介绍了软件测试的流程，以及测试流程过程中需要提交哪些文档；然后从各个需要提交的模板来分析测试文档该如何编写。通过学习本章，应对测试文档有清晰的了解，

并学会测试文档的写作方法。

习　　题

1. 软件测试需要哪些文档？
2. 软件测试计划需要遵循哪些文档来编写？
3. 如何编写测试用例？
4. 如何编写测试报告？

第 7 章　黑盒测试法案例分析

7.1　黑盒测试工具分类介绍

随着人们对测试的重视，手工测试很难满足软件测试的需求。因此，市场上很快出现了各种自动化的测试工具。作为测试人员，必须掌握这些工具的使用，才能够很好地完成测试工作。

软件测试工具尽管种类繁多，但是总的来说可以按照其测试方法的不同分为两大类，一类是黑盒测试工具，另一类是白盒测试工具。本章重点介绍黑盒测试工具——IBM Rational Function Tester，第 8 章将介绍一些常用的白盒测试工具。

一般来说，黑盒测试工具包括功能测试工具和系统测试工具两大类。其中功能测试工具能够帮助工作人员验证软件是否能够达到预期的功能并正常运行。这类工具的典型代表有 WinRunner、QTP、Rational Function Tester 和 Rational Robot 等。系统测试工具是预测系统行为和性能的自动化测试工具，能够及时发现系统性能瓶颈，并为系统优化提供参考。这类工具有代表性的是 LoadRunner 和 WAST（Web Application Stress Test）。

1. 负载压力测试工具

这类测试工具的主要目的是度量应用系统的可扩展性和性能，是一种预测系统行为和性能的自动化测试工具。在实施并发负载的过程中，通过实时性能监测来确认和查找问题，并针对所发现的问题对系统性能进行优化，确保应用的成功部署。负载压力测试工具能够对整个企业架构进行测试，通过这些测试，企业能最大限度地缩短测试时间，优化性能和加速应用系统的发布周期。表 7.1 对主要的性能测试工具作了概要介绍。

表 7.1　性能测试工具分类介绍

工具名称	来源	类型	功能概要
LoadRunner	Mercury 公司	性能与负载压力	LoadRunner 是一种预测系统行为和性能的工业标准级负载测试工具。通过模拟上千万用户实施并发负载及实时性能监测的方式来确认和查找问题，LoadRunner 能够对整个企业架构进行测试。通过使用 LoadRunner，企业能最大限度地缩短测试时间，优化性能和加速应用系统的发布周期。LoadRunner 是一种适用于各种体系架构的自动负载测试工具，它能预测系统行为并优化系统性能。LoadRunner 的测试对象是整个企业的系统，它通过模拟实际用户的操作行为和实行实时性能监测，来帮助测试人员更快地查找和发现问题。此外，它还能支持广泛的协议和技术，为特殊环境提供特殊的解决方案

续表

工具名称	来源	类型	功能概要
SilkPerformer	Segue 公司	负载压力测试	SilkPerformer 是一种在工业领域最高级的企业级负载测试工具。它可以模仿成千上万的用户工作在多协议和多计算的环境下。利用 SilkPerformer 可以在企业电子商务应用部署前预测其性能,不论其大小和复杂性。可视的用户化界面、实时的性能监控和强大的管理报告可以帮助用户迅速地解决问题。SilkPerformer 提供了在广泛的、多样的状况下对电子商务应用进行弹性负载测试的能力,通过 TrueScale 技术,SilkPerformer 可以从一台单独的计算机上模拟成千上万的并发用户,在使用最小限度的硬件资源的情况下,提供所需的可视化结果确认的功能。在独立的负载测试中,SilkPerformer 允许用户在多协议和多计算环境下工作,并可以精确地模拟浏览器与 Web 应用的交互作用。SilkPerformer 的 TrueLog 技术提供了完全可视化的原因分析技术。通过这种技术可以对测试过程中用户产生和接收的数据进行可视化处理,包括全部嵌入的对象和协议头信息,从而进行可视化分析,甚至在应用出现错误时都可以进行问题定位与分析
Rational Performance Tester	IBM 公司	负载和性能测试	自动负载和性能测试工具,用于开发团队在部署基于 Web 的应用程序前验证其可扩展性和可靠性。它提供了可视化编辑器,使新的测试人员可以简单地使用。为需要高级分析和自定义选项的专家级测试人员提供了对丰富的测试详细信息的访问能力,并支持自定义 Java 代码插入。自动检测和处理可变数据,以简化数据驱动的测试。提供有关性能、吞吐量和服务器资源的实时报告,以便及时发现系统的瓶颈。可以在 Linux 和 Windows 上进行测试录制和修改
QALoad	Compuware 公司	负载压力测试	QALoad 是客户/服务器系统、企业资源配置(ERP)和电子商务应用的自动化负载测试工具。QALoad 是 QACenter 性能版的一部分,它通过可重复的、真实的测试能够彻底地度量应用的可扩展性和性能。QACenter 汇集完整的跨企业的自动测试产品,专为提高软件质量而设计。QACenter 可以在整个开发生命周期、跨越多种平台、自动执行测试任务。在投产准备时期,QALoad 可以模拟成百上千的用户并发执行关键业务而完成对应用程序的测试,并针对所发现的问题对系统性能进行优化,确保应用的成功部署。预测系统性能,通过重复测试寻找瓶颈问题,从控制中心管理全局负载测试,验证应用的可扩展性,快速创建仿真的负载测试

续表

工具名称	来源	类型	功能概要
WebLoad	Radview 公司	性能测试、压力测试	WebLoad 专为测试在大量用户访问下的 Web 应用性能而设计。其控制中心运行在 Windows 2000,XP 和 2003 操作系统上,负载发生模块(load machine)可以运行在 Windows、Solaris 和 Linux 操作系统上。模拟出来的用户流量可支持.NET 和 J2EE 两种环境。WebLoad 的测试脚本采用 JavaScript 脚本语言实现,支持在 DOM(Document Object Model)的基础上将测试单元组织成树形结构,对 Web 应用进行遍历或者选择性测试。WebLoad 还可以录制用户访问 Web 应用的操作过程,自动生成测试脚本,也可以使用脚本编辑器手工编辑或者修改脚本。WebLoad 的专利技术可以让用户为系统设定最低可接受性能门限值,并让 WebLoad 采用自增用户数的循环测试方式进行测试,这样 WebLoad 就可以自动测得系统的最大用户容量。WebLoad 不仅能够测试 Web 性能,还能通过直观的图形用户界面直接连接到数据库,测试数据库性能。还可以测试多种 Internet 协议,如 FTP、Telnet、SMTP 和 POP 等的性能。WebLoad 还可以模拟 DDoS 攻击。它可以模拟诸如 Tfn、Tfn2K、Trinoo、Smurf、Flitz、Carko、Omega3、Plague、TCP Flood(SYN、ACK)、UDP Flood、ICMP Flood(Ping、Host-Unreachable)等攻击。通过模拟 DDoS 攻击可以测试 Web 系统在面临 DDoS 攻击的时候可用性和反应时间的受影响情况。同时 WebLoad 提供有关 DDoS 攻击测试的详细报告。它可以帮助用户分析系统漏洞和弱点,为用户加固系统提供依据
WebLoad Analyzer	Radview 公司	性能测试	WebLoad Analyzer 用来发现、诊断并定位 Web 应用性能问题。WebLoad Analyzer 使用一个安装于服务器的探针程序搜集所需的应用进程以及操作系统信息。用户可以定制探针程序的搜集行为。它支持多种操作系统和应用服务。WebLoad Analyzer 同时分析外部的性能测试数据和内部监视数据。它可以监视多种 Web 应用服务、操作系统和数据库,并能将数据自动相关和同步,帮助用户分析定位性能问题。WebLoad Analyzer 搜集 Web 应用各层的性能数据,使用专利技术分析数据,定位问题,并且将重要信息发送回控制中心,用户可以定制使用 E-mail、弹出页面或者 Snmp-Trap 的方式告警。WebLoad Analyzer 不仅能对问题发出告警,还能深入分析问题,找出问题根源,如找到导致问题的 Java 容器、组件、类或者方法等
PureLoad	Minq 公司	负载压力测试	PureLoad 正是一款基于 Java 开发的网络负载压力测试工具,它的 Script 代码完全使用 XML,所以,这些代码的编写很简单,可以测试各种 C/S 程序,如 SMTP Server 等。它的测试报表包含文字和图形并可以输出为 HTML 文件。由于它是基于 Java 的软件,所以,可以通过 Java Beans API 来增强软件功能

续表

工具名称	来源	类型	功 能 概 要
JMeter	开源组织	压力测试和性能测试	它最初被设计用于 Web 应用测试，但后来扩展到其他测试领域。Apache JMeter 可以用于对静态的和动态的资源（文件、Servlet、Perl 脚本、Java 对象、数据库和查询和 FTP 服务器等）的性能进行测试。它可以用于对服务器、网络或对象模拟繁重的负载来测试它们的强度或分析不同压力类型下的整体性能。可以使用它做性能的图形分析，或在大并发负载下测试服务器/脚本/对象。Apache JMeter 的特性包括：能够对 HTTP 和 FTP 服务器进行压力和性能测试，也可以对任何数据库进行同样的测试（通过 JDBC）；完全的可移植性和 100%纯 Java；完全 Swing 和轻量组件支持（预编译的 JAR 使用 javax. swing. * ）包；完全多线程框架允许通过多个线程并发取样和通过单独的线程组对不同的功能同时取样；精心的 GUI 设计允许快速操作和更精确的计时；缓存和离线分析/回放测试结果；高可扩展性：可链接的取样器允许无限制的测试能力；各种负载统计表和可链接的计时器可供选择。数据分析和可视化插件提供了很好的可扩展性以及个性化；具有提供动态输入到测试的功能（包括 JavaScript）；支持脚本变成的取样器（在 1.9.2 及以上版本中支持 BeanShell）
OpenSTA	开源组织	性能测试	OpenSTA 是专用于 B/S 结构的、免费的性能测试工具。它的优点除了免费和源代码开放外，还能对测试脚本进行录制，并按指定的语法进行编辑。测试工程师在录制完测试脚本后，只需要了解该脚本语言的特定语法知识，就可以对测试脚本进行编辑，以便再次执行性能测试时获得所需要的参数，之后进行特定的性能指标分析。OpenSTA 以最简单的方式让大家对性能测试的原理有较深的了解，其较为丰富的图形化测试结果大大提高了测试报告的可阅读性。OpenSTA 是基于 Common Object Request Broker Architecture（CORBA）的结构体系。它虚拟一个 proxy，使用其专用的脚本控制语言，记录通过 proxy 的一切 HTTP/HTTPS traffic。测试工程师通过分析 OpenSTA 的性能指标收集器收集的各项性能指标以及 HTTP 数据，对被测试系统的性能进行分析
Microsoft Web Application Stress Tool	微软公司	压力性能测试	使用集中压力测试对每个单独的组件进行压力测试后，应对包含所有组件和支持服务的整个应用程序进行压力测试。集中压力测试主要关注与其他服务、进程以及数据结构（来自内部组件和其他外部应用程序服务）的交互。集中测试从最基础的功能测试开始。测试人员需要知道编码路径和用户方案、了解用户试图做什么以及确定用户运用应用程序的所有方式。使用真实环境测试在隔离的、受保护的测试环境中可靠的软件，在真实环境的部署中可能并不可靠。虽然隔离测试在早期的可靠性测试进程中是有用的，但真实的测试环境才能确保并行应用程序不会彼此干扰。这种测试经常发现与其他应用程序之间意外地导致失败的交互。使用随机破坏测试对可靠性进行测试的一个最简单的方法是使用随机输入。这种类型的测试通过提供虚假的、不合逻辑的输入，努力使应用程序发生故障或挂起。输入可以是键盘或鼠标事件、程序消息流、Web 页、数据缓存或任何其他可强制进入应用程序的输入情况。应该使用随机破坏测试对重要的错误路径进行测试，并公开软件中的错误。这种测试通过强制失败以便观察返回的错误处理来改进代码质量

2. 功能测试工具

功能测试工具通过自动录制、检测和回放用户的应用操作，将被测系统的输出记录同预先给定的标准结果比较，能够有效地帮助测试人员对复杂的企业级应用的不同发布版本的功能进行测试，提高测试人员的工作效率和质量。其主要目的是检测应用程序是否能够达到预期的功能并正常运行。表7.2对主要的功能测试工具作了概要介绍。

表7.2 功能测试工具分类介绍

工具名称	来源	类型	功能概要
WinRunner	Mercury公司	功能测试	WinRunner最主要的功能是自动重复执行某一固定的测试过程，它以脚本的形式记录手工测试的一系列操作，在环境相同的情况下重放，检查其在相同的环境中有无异常的现象或与实际结果不符的地方，可以减少由于人为因素造成结果错误，同时也可以节省测试人员大量的测试时间和精力来做别的事情。功能模块主要包括GUI map、检查点、TSL脚本编程、批量测试和数据驱动等几部分
QuickTest Pro	Mercury公司	功能测试和回归测试	QTP是一个B/S系统的自动化功能测试工具，可以覆盖绝大多数的软件开发技术，简单高效，并具备测试用例可重用的特点。QuickTest Pro是一款先进的自动化测试解决方案，用于创建功能和回归测试。它自动捕获、验证和重放用户的交互行为
SilkTest	Segue公司	功能测试和回归测试	SilkTest是面向Web应用、Java应用和传统的C/S应用，进行自动化的功能测试和回归测试的工具。它提供了用于测试的创建和定制的工作流设置、测试计划和管理、直接的数据库访问及校验等功能，使用户能够高效率地进行软件自动化测试。 为提高测试效率，SilkTest提供多种手段来提高测试的自动化程度，包括测试脚本的生成、测试数据的组织、测试过程的自动化和测试结果的分析等方面。在测试脚本的生成过程中，SilkTest通过动态录制技术录制用户的操作过程，快速生成测试脚本。在测试过程中，SilkTest还提供了独有的恢复系统（Recovery System），允许测试可在24×7×365全天候无人看管条件下运行。在测试过程中一些错误导致被测应用崩溃时，错误可被发现并记录下来，之后，被测应用可以被恢复到它原来的基本状态，以便进行下一个测试用例的测试
Robot	IBM公司	功能测试和回归测试、集成测试	IBM Rational Robot是业界最顶尖的功能测试工具，它甚至可以在测试人员学习高级脚本技术之前帮助其进行成功的测试。它集成在测试人员的桌面IBM Rational TestManager上，在这里测试人员可以计划、组织、执行、管理和报告所有测试活动，包括手动测试报告。IBM Rational Robot是一种可扩展的、灵活的功能测试工具，经验丰富的测试人员可以用它来修改测试脚本，改进测试的深度。IBM Rational Robot自动记录所有测试结果，并在测试日志查看器中对这些结果进行颜色编码，以便进行快速可视分析。多种IDE和语言支持Java环境，以及Microsoft Visual Studio. NET，HTML、XML和DHTML应用程序，Oracle Developer/2000，Visual Basic应用程序，PowerBuilder应用程序等

续表

工具名称	来源	类型	功 能 概 要
Manual Tester	IBM 公司	手工测试自动化工具	IBM Rational Manual Tester 是一个易于使用的自动化工具,用来加速和提高手动测试的准确度。适用于使用自动化和手工测试方法的团队,同样也适用于那些没有测试自动化工具的团队。关键能力包括:一个进行测试验证的组件化的"构建阻塞"方法;简化使用单点更新的测试维护;开发健壮的、易读的手工测试的 RichText 编辑;批量导入 Microsoft Word 和 Excel 的手工测试文档;提高手工测试执行的准确度和速度的辅助数据入口;在测试执行期间的辅助数据对比;支持分布式团队
Functional Tester	IBM 公司	功能测试和回归测试	Rational Functional Tester 是一个面向对象的自动测试工具,它可用于测试多种应用程序。可以通过记录对应用程序的测试来快速地生成脚本,并且可以测试应用程序中的任意对象,包括对象的属性和数据。Rational Functional Tester 提供一个选择脚本语言和开发环境的机会——Eclipse 框架中的 Java 或者 Microsoft Visual Studio . NET 开发环境中的 Microsoft Visual Basic . NET。将 Rational Functional Tester 集成到开发平台中可以把开发和测试带到一个新的效率级别上。进行安装后,Rational Functional Tester 就成为用户本地开发环境中的一个无缝部分。Rational Functional Tester 提供记录和回放功能,并存储 Java 或 . NET 源代码的记录脚本。允许用户实际上不做任何工作就能创建可重复的测试脚本,也允许用户使用本地的开发语言来增强脚本以满足具体的需求
Logiscope	Telelogic 公司	功能测试	Logiscope 是一种软件质量保证(QA)工具,它可以通过自动进行代码检查和对容易出错的模块的鉴定与检测来帮助扩大测试范围,从而达到保证质量和完成软件测试的目的。可自定义的软件测试功能可帮助开发人员在软件开发过程中及早发现缺陷,这样开发人员就可以做到按时交付,将费用控制在预算内,同时又可以提高软件质量。在软件开发生命周期的早期排除错误对于维护软件开发标准是至关重要的,这样开发人员就可以满足需求、构建可靠产品,并最大限度地缩短将产品推向市场的时间。Logiscope 可以鉴定出很可能包含缺陷的模块,向开发人员说明有缺陷的结构,并提供改进建议
QACenter	Compuware 公司	功能测试、性能测试和回归测试等	QACenter 帮助所有的测试人员创建一个快速、可重用的测试过程。这些测试工具自动帮助管理测试过程,快速分析和调试程序,包括针对回归、强度、单元、并发、集成、移植、容量和负载,建立测试用例,自动执行测试和产生文档结果。QACenter 主要包括以下几个模块:QARun,应用的功能测试工具;QALoad,强负载下应用的性能测试工具;QADirector,测试的组织设计和创建以及管理工具;TrackRecord,集成的缺陷跟踪管理工具;EcoTools,高层次的性能监测工具

续表

工具名称	来源	类型	功 能 概 要
TestPartner	Compuware 公司	功能测试	TestPartner 是 Compuware 公司的一个自动化测试工具，它能提高复杂应用的功能测试效率，对 Microsoft 平台、Java 平台和 Web 平台的应用都适用。使用 TestPartner 的通用的、层级化的发方法，测试人员有没有编程经验都可以使用 Visual Navigator 快速地录制和回放测试脚本。TestPartner 按树形结构记录和展示测试。这些图形可以清晰地验证 Web 应用的测试路径、点击对象以及输入的数据，提供可视化的高级脚本语言表示法。TestPartner 的特色是多层次开发测试脚本。开发人员和测试技术人员可以更充分地利用 VBA 脚本编制和调试功能，创建先进的测试用例。没有编程知识的测试人员也能够使用 Visual Navigator 建立已录制脚本的可视化图示。层级化方法还在保留丰富的测试功能时，缩短了脚本编写的学习曲线。用其他工具测试.NET 环境时测试人员会觉得对复杂应用难于掌控。而使用 TestPartner，测试人员可以深挖到.NET 对象、展示客户属性以及那些通过 VB 实现的功能。TestPartner 能够测试基于组件的应用，包括测试在客户端或在服务器端的 GUI 和非 GUI 的 COM 组件。TestPartner 是唯一的，可以在测试服务器端 COM 对象时，同时测试客户端已经运行的 COM 对象和测试工具。TestPartner 能够与 DevPartner 和 QACenter 产品线的开发效能管理工具、缺陷跟踪工具、测试管理工具和负载测试工具集成。TestPartner 的集成能力为分布式应用开发和测试的提速提供了最全面的、端到端的解决方案。它改善了开发和测试团队之间的沟通，使他们能够更紧密地工作，在开发生命周期尽早找到和解决问题
E-TEST Suite	Empirix 公司	功能测试、压力测试	E-TEST Suite 测试软件是当前优秀的易于使用、并能够和被测试应用无缝结合的 Web 应用测试工具。该产品由 3 部分组成：e-TESTER、e-LOAD 和 e-MONITOR，这 3 种工具分别适用于应用功能测试、压力测试以及应用监控，每一部分的功能相互独立，测试过程中又可以彼此协同，从多方面保障了 Web 应用的成功。e-TESTER 能自动测试每星期甚至每天都在变化着的 Web 应用程序。它可以记录测试过程中所访问的每一页面上的所有对象，并以图形化的方式呈现，从而任何差异都可以被凸显出来，以此来测试 Web 应用的功能。 Web 应用程序的主要好处之一是允许大量用户同时访问。相应地，开发人员关心应用是否具有良好的性能，以支持大规模的访问。e-LOAD 提供了非常出色的压力测试解决方案，它在开发的过程中创造了一个仿真环境，能够模拟真实用户访问 Web 应用，提供全面的应用性能统计信息。为了保证 Web 应用能够为用户提供不间断服务，维护人员应当监控应用的运行状况。e-MONITOR 可以 7×24 小时地执行监控工作，允许使用者设置各种报警方式及时报告应用的问题，以便管理人员迅速做出反应

续表

工具名称	来源	类型	功能概要
WebFT	Radview 公司	功能测试	WebFT 帮助用户对 Web 系统进行快速、有效的功能性测试。它是模拟单用户对网站进行功能测试的。WebFT 支持 3 个测试级别：全局、页面和对象，可以测试系统或者页面的全部功能，也可以深入细致地测试页面上某个对象的功能，如 HTML 页面的某个属性，某个嵌入的 Java 对象或者 ActiveX 控件。WebFT 测试脚本与 WebLoad 的测试脚本完全一样，也是使用 JavaScript 语言写成的，也能够自动生成。因此 WebFT 使用的脚本也可以在 WebLoad 中使用
Jameleon	开源组织	功能测试	Jameleon 是一个自动化测试工具，用来测试各种各样的应用程序，所以它被设计成插件模式。为了使整个测试过程变得简单，Jameleon 提供了一个 GUI，因此 Jameleon 实现了一个 Swing 插件
Abbot Java GUI Testing Framework	开源组织	功能测试、GUI 测试	Abbot Java GUI Testing Framework 是用来对 Java 的图形界面应用程序进行功能和单元测试的一个简单的框架。其主要功能包括模拟用户行为以及检查组件状态，测试过程会被记录下来并可以进行回放

3. 测试辅助工具

测试辅助工具本身并不执行测试，例如它们可以生成测试数据，为测试提供数据准备。

TestDataBuilder 是一个采用 Java 编写的、完全开源、免费的测试数据生成工具，软件遵循 GPL 协议。TestDataBuilder 可以帮助程序开发或测试人员自动生成数据库表中的测试数据，并且具有期望的值分布和列间相关性，可以通过配置工具配置数据生成的规则，并且有一个自动配置引擎。TestDataBuilder 可以根据已经存在的数据库自动生成配置文件，可以支持 JDBC 所支持的所有数据库类型，用户也可以自己开发新数据类型。TestDataBuilder 支持中文和英文两个语言版本，附带一个查询控制台，可以处理常规 SQL 查询操作，生成的测试数据直接插入到数据表中，也可以以 Insert 语句形式记录在文件中。

DataFactory（数据工厂）是 Quest 公司的测试辅助软件。顾名思义，数据工厂是生产数据的，该工具的主要应用领域是性能测试中的大数据量测试，也就是性能测试的数据准备阶段。它是一种快速产生测试数据的工具，具有良好的用户界面，能建模复杂数据关系，允许开发人员和 QA 很容易地产生百万行有意义的、正确的测试数据库。该工具支持 DB2、Oracle、Sybase 和 SQL Server 数据库，支持 ODBC 连接方式，但无法直接使用 MySQL 数据库。DataFactory 首先读取一个数据库方案，用户随后用鼠标操作即可产生一个数据库。

下面详细介绍功能测试工具 Rational Function Tester（7.2 节）以及系统测试工具 LoadRunner（第 9 章），以使读者更好地了解黑盒测试工具的性能特点。

7.2 IBM Rational Function Tester 测试工具

Rational Functional Tester 是一个面向对象的自动测试工具，能够测试各种应用程序。Rational Functional Tester 通过录制一个应用程序的测试过程，可以方便地得到测试脚本。

还可以测试应用程序之中的任何对象，包括对象的属性和数据。

Rational Functional Tester 可以提供一个编写脚本语言的机会和两种开发环境：Eclipse 框架中的 Java 或者 Microsoft Visual Studio 开发系统中的 Microsoft Visual Basic .NET。这意味着，无论开发小组的成员使用什么样的语言或者平台，都应该能够将它们与 Rational Functional Tester 集成起来，并且在开发自动化测试的时候能够利用它们的一些功能。

Rational Functional Tester 针对 Java、.NET 的对象技术和 Web 应用程序进行录制、回放等测试操作，为测试者提供测试自动化帮助。Rational Functional Tester 会为被测的应用程序自动创建测试对象。对象中包含了对每个对象的识别属性。在对象中更新记录信息时，任何使用了该对象的脚本会共享更新的信息，降低了维护的成本及整个脚本开发的复杂度。

Rational Functional Tester 还提供快速的方法向脚本中添加更多的对象。它列出应用程序中涉及的测试对象，不论它们当前是否可视，都可以通过依据现有地图或按需添加对象来创建新的测试对象地图。

在记录过程中可以将验证点插入到脚本中，方便用户了解在被测应用程序建立过程中，测试对象的状态。验证点可以获取对象信息，并在基本数据文件中存储，该文件中的信息将会成为建立过程中测试对象的期望状态。在执行完测试之后，还可以使用验证点比较器(verification point comparator)进行分析，将对象的执行状态与期望状态进行比较，如果对象的行为发生变化了，就更新基线，即更新对象的期望状态。

7.2.1 工具安装及基本使用

用户可以使用安装光盘安装 Rational Functional Tester。首先，打开安装光盘，双击 launchpad.exe，启动 Rational Software Development Platform 安装界面进行智能安装，如图 7.1 所示。

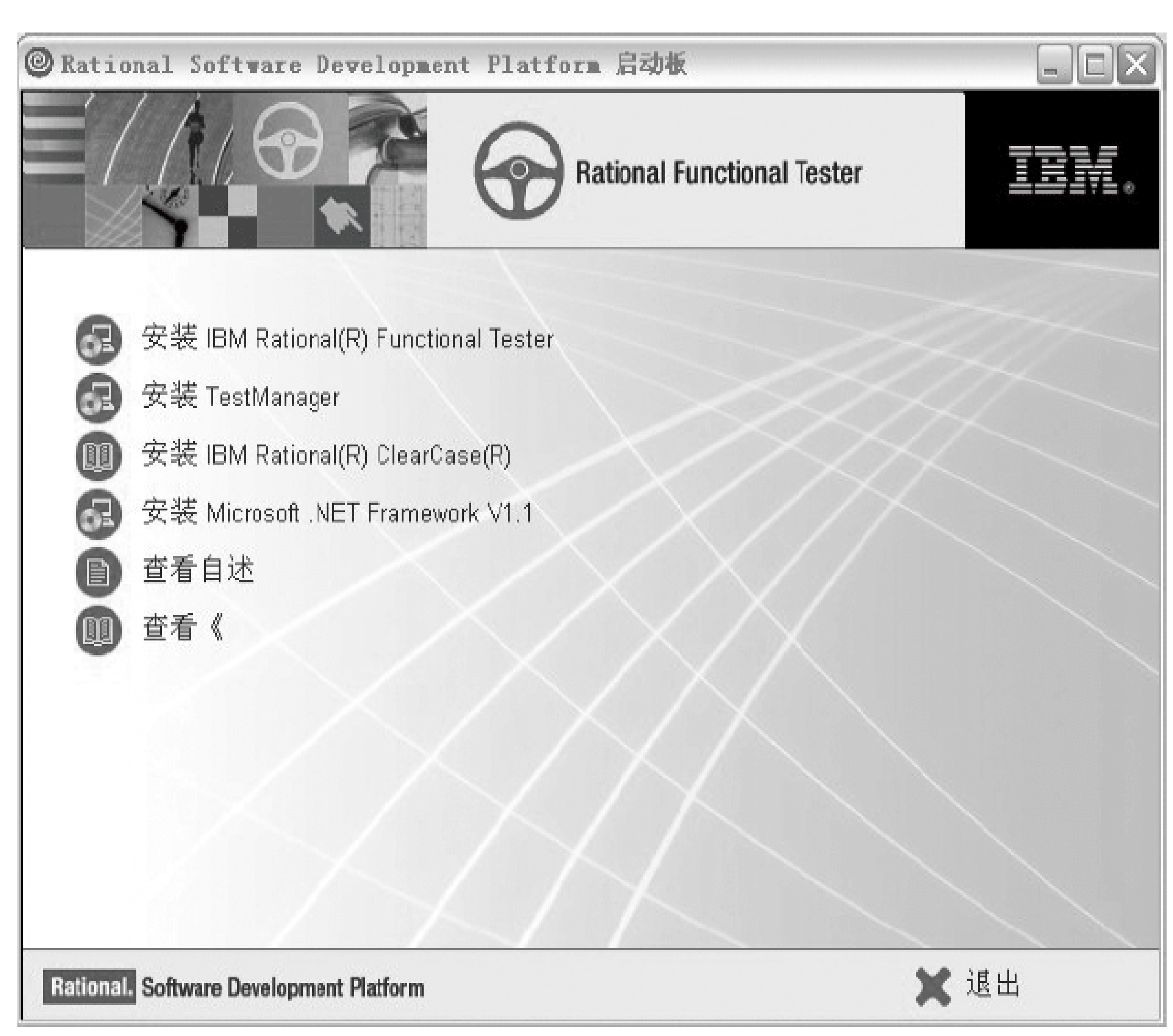


图 7.1 安装启动界面

该界面提供IBM Rational系列软件的安装向导。选择“安装IBM Rational Functional Tester”进行安装。接下来安装程序会根据用户的系统选择合适版本的软件进行安装,本书采用的是Windows系统。在此基础上,安装向导会让用户选择需要安装的产品,我们选择安装Java脚本的编制,当然系统也提供了VB.NET脚本的编制方式。因为Functional Tester会自动录制脚本,这里如果选择Java脚本,Functional Tester会用Java脚本语言自动录制脚本。用户可以根据自己的需要选择所需组件,如图7.2所示。之后的安装过程读者可按系统提示进行。



图7.2　选择所需组件

另外,用户也可以使用在线方式安装。登录http://www.ibm.com/us/en/选择最新版本软件的安装。这里需要先注册为IBM公司的用户,IBM网站会提供30天的免费试用软件。

安装完成后,读者可按以下4部分设置,使用Functional Tester。

1. Rational Functional Tester 的启动

(1) 启动:选择“开始”→“程序”→IBM Rational→Rational Functional Tester→Java Scripting命令。

(2) 输入工作空间路径,例如D:\workspace。

工作空间可以根据用户需要自行创建。在工作空间中将会记录测试脚本和测试日志等文档。工作空间设置如图7.3所示。

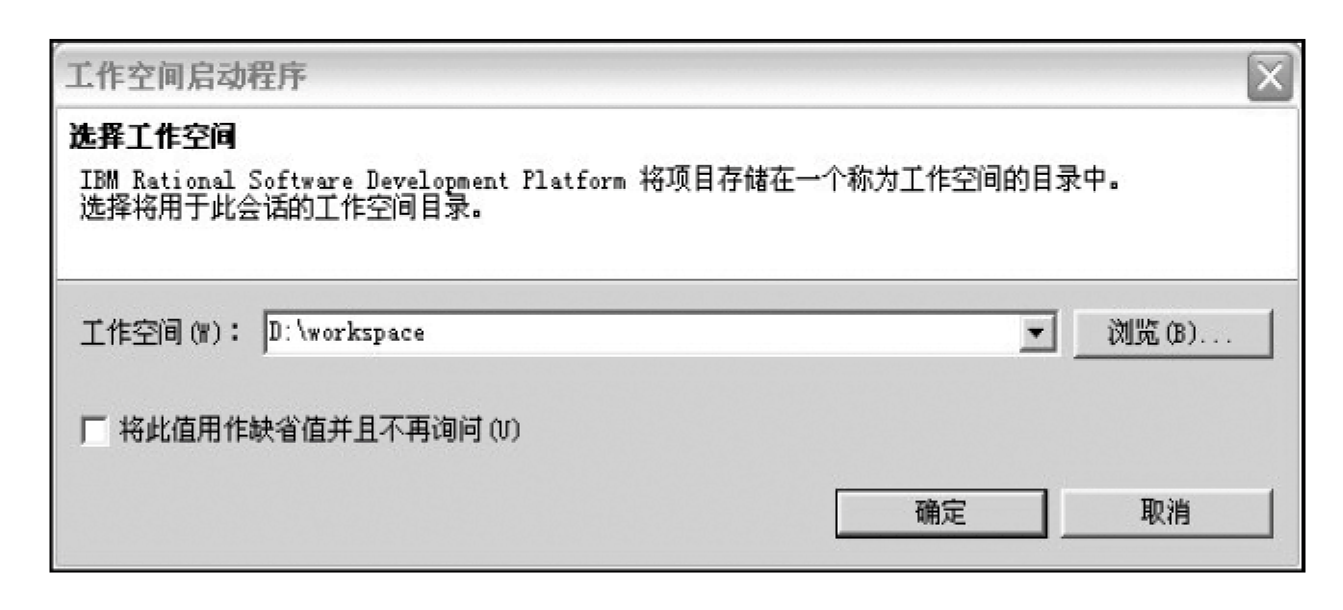


图7.3　设置工作空间

2. 设置日志记录选项

(1) 选择菜单命令"窗口"→"首选项"。

(2) 在左侧窗口的树形图中选择:"功能测试"→"回放"→"日志记录",如图 7.4 所示。

(3) 可以设置日志记录的相关内容,这里采用 HTML 形式来记录日志,也就是将日志记录到 HTML 文件中。

(4) 确认"日志类型"下拉列表框右边的"使用缺省值"复选框已被选中,并且 HTML 出现(变灰)在下拉列表框中。

Function Test 参数设置如图 7.4 所示。



图 7.4　设置 Function Test 参数

3. 创建 Functional Test 项目

运行菜单命令"文件"→"新建"→"Functional Test 项目",如图 7.5 所示。

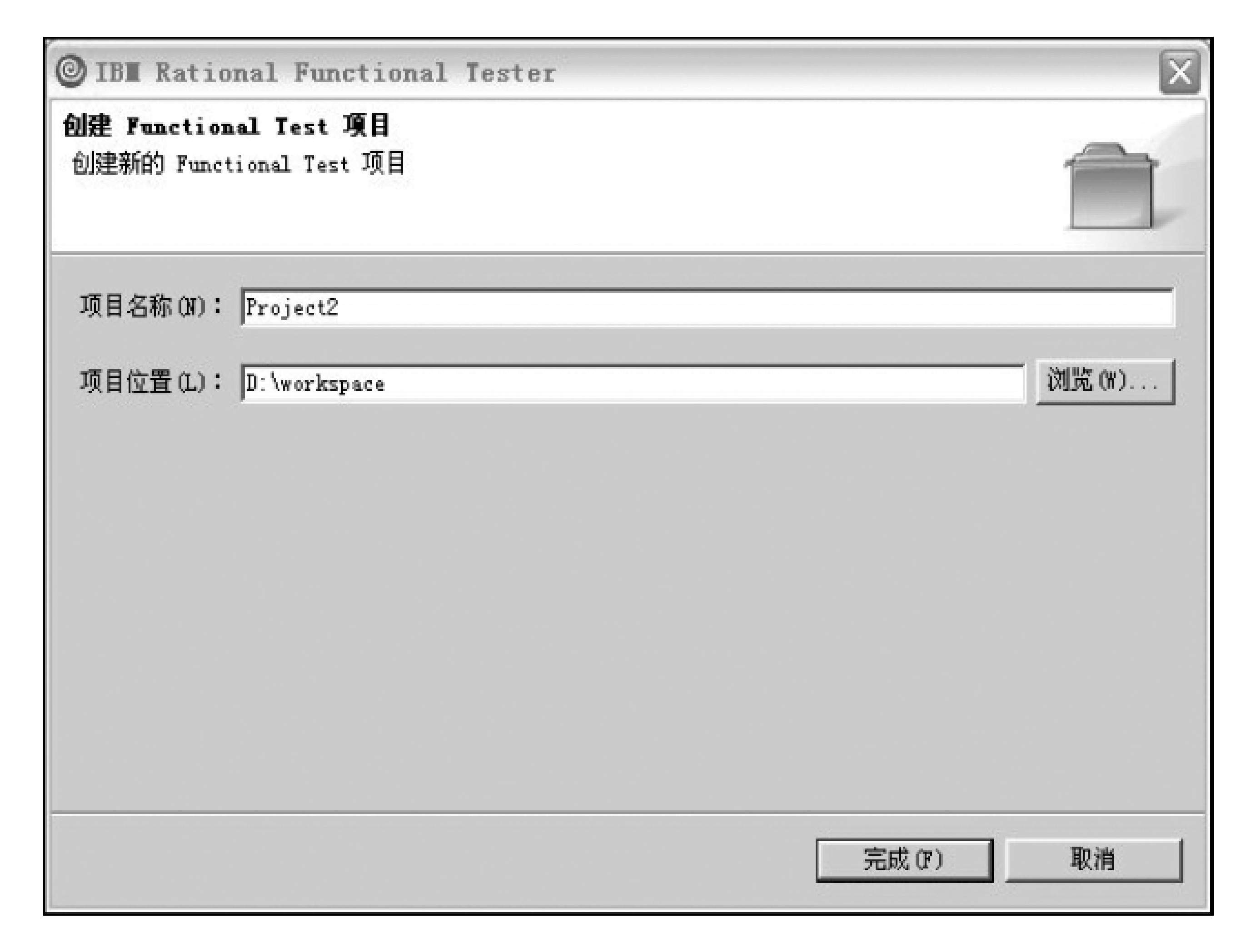


图 7.5　创建项目

(1) 在“项目名称”文本框输入项目名，注意项目名中不要加任何空格。

(2) 在“项目位置”文本框输入项目要存放的位置，Functional Test 会自动创建这个目录，一般默认为 Functional Test 的工作空间。

(3) 如果有源控制选项可用，则不要选择将项目添加到源控制中。

(4) 如果有关联项目选项可用，则不要选择将 Functional Test 项目与当前 Rational 项目相关联。

(5) 单击“完成”按钮。

4. Rational Functional Tester 主界面

启动 Rational Functional Tester 时，会看到带有 6 个主要组件。下面分别进行介绍。

1) 主菜单

Rational Functional Tester 主菜单如图 7.6 所示。



图 7.6　主菜单

2) 工具栏

Rational Functional Tester 工具栏如图 7.7 所示。

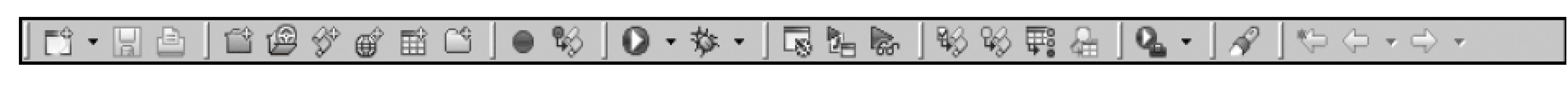


图 7.7　工具栏

工具栏中包含以下图标。

新建：显示适当的对话框来创建许多项中的一个或录制 Functional Test 脚本。单击右侧的三角形以显示要创建的可能项列表。

创建 Functional Test 项目：显示出一个对话框，让用户在 Functional Test 中生成新工程。

连接到现有的 Functional Test 项目：显示出一个让用户连接到现有工程的对话框。

创建空 Functional Test 脚本：显示一个可以用来手动添加 Java 代码脚本的对话框。

创建测试对象图：显示一个向工程添加新测试对象地图的对话框。

创建测试数据池：显示一个创建新的测试数据库的对话框。

创建测试文件夹：显示一个为工程或现有文件夹创建新文件夹的对话框。

记录 Functional Test 脚本：显示一个输入关于新脚本的信息并开始记录的对话框。

将记录插入活动的 Functional Test 脚本：在当前脚本的光标位置开始记录，它能启动应用程序，插入验证点，并添加脚本支持功能。

配置应用程序进行测试：显示 Application Configuration 工具，可以添加并编辑要测试的 Java 和 HTML 应用程序的配置信息(例如名称、路径和其他用于开始并执行应用程序的信息)。

启动环境进行测试：显示一个用来启动 Java 环境和浏览器及配置 JRE 和浏览器的对话框。

打开测试对象检测器：显示 TestObject Inspector 工具，显示测试对象信息，如父层次、继承层次、测试对象属性、无值属性和方法信息。

将验证点插入活动的 Functional Test 脚本：显示 Verification Point and Action Wizard 的 Select an Object 页，可在要测试的应用程序中选择对象。

将测试对象插入活动的 Functional Test 脚本：显示一个选择测试对象添加到测试对象地图和脚本中的对话框。

将数据驱动的命令插入活动的 Functional Test 脚本：显示出 Datapool Population Wizard 的 Data Drive Actions 页，可选择被测应用程序中的对象来数据驱动应用程序。

查找字面值并替换为数据池引用：用测试脚本中的数据库参考代替文字值，可向现有的测试脚本中添加现实数据。

运行 Functional Test 脚本：运行 Functional Test 脚本。单击右侧的小三角形可显示运行命令列表。

调试 Functional Test 脚本：启动当前脚本并显示 Debug Perspective(在脚本调试时提供信息)。单击该按钮以在当前脚本的方法 Main 中开始调试。单击右侧的小三角形以显示调试命令列表。

外部工具：使用户可以配置非工作台一部分的外部工具。单击右侧的小三角形显示选项列表。

3) 项目视图

Rational Functional Test 项目视图如图 7.8 所示。

Functional Test 项目视图在 Test Perspective 窗口的左窗格中，为每个测试工程列出测试资源，包括以下内容：

文件夹。

脚本。

共享的测试对象地图。

日志文件夹。

日志。

Java 文件。

4) 脚本编辑器

Rational Functional Test 的脚本编辑器如图 7.9 所示。

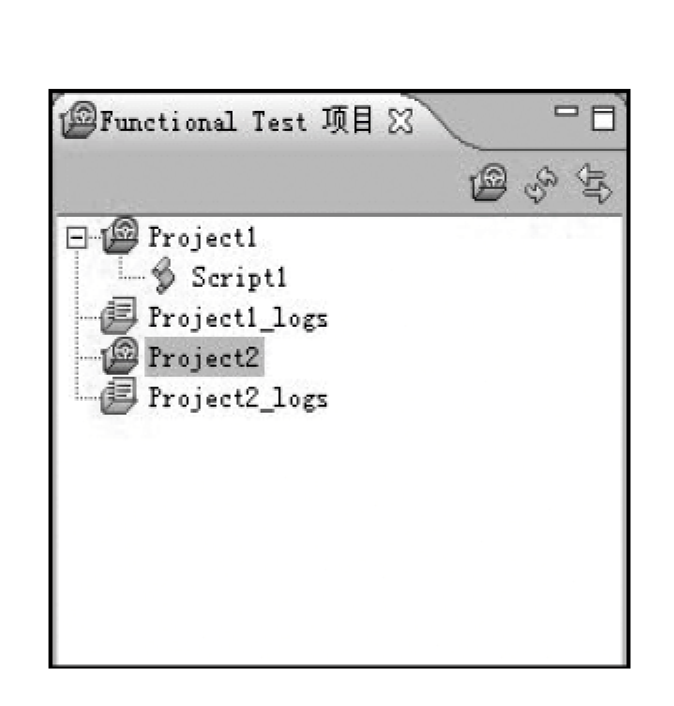


图 7.8　项目视图

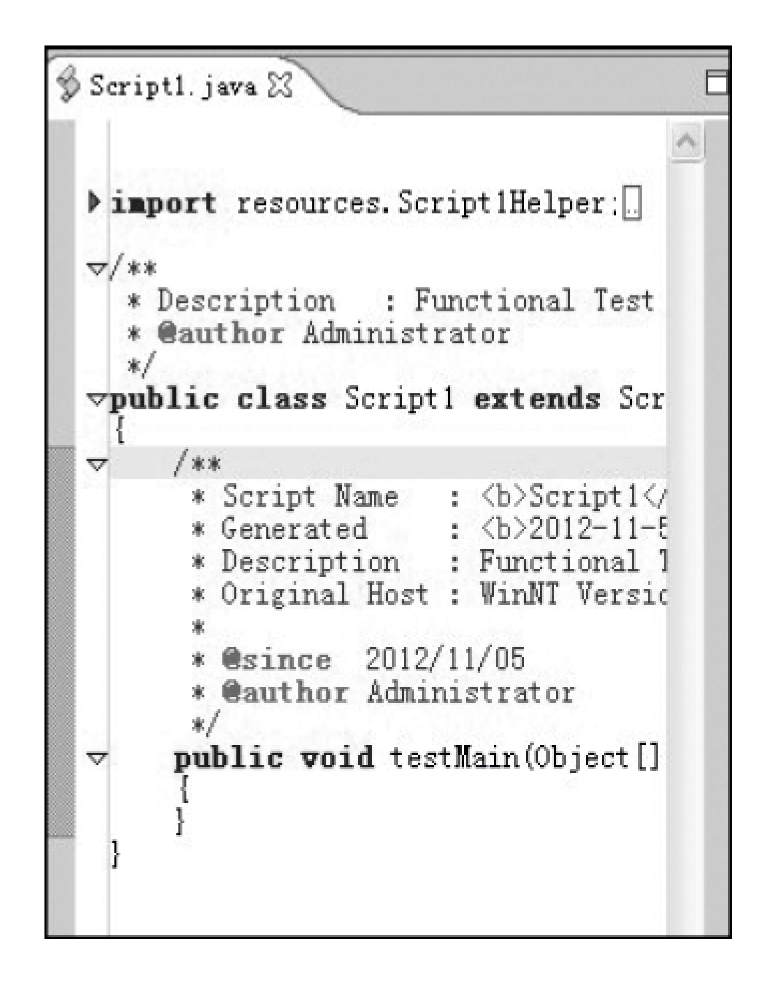


图 7.9　脚本编辑器

5）脚本资源管理器

Rational Functional Test 的脚本资源管理器如图 7.10 所示。

6）控制栏、任务栏和状态栏。

Rational Functional Test 的控制栏、任务栏和状态栏一般出现在测试脚本下方，如图 7.11 所示。



图 7.10 脚本资源管理器

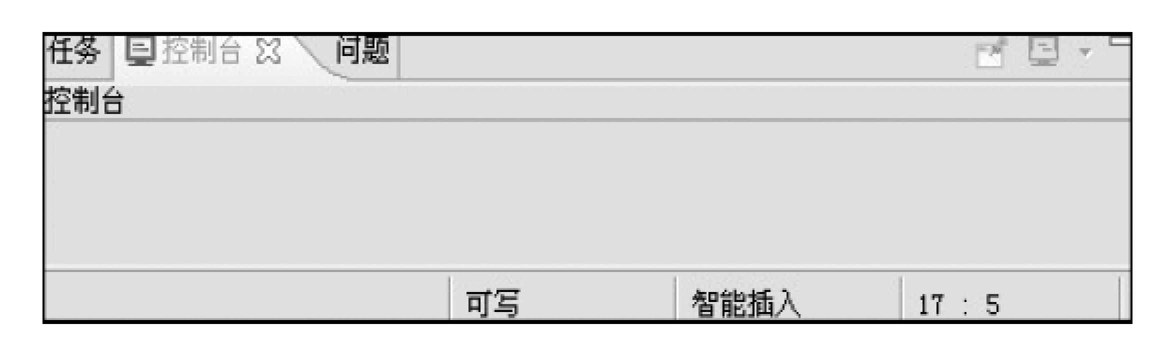


图 7.11 控制栏、任务栏和状态栏

7.2.2 脚本录制与回放

Function Tester 可针对测试应用程序记录功能测试脚本，然后在应用程序或者应用程序新构建版本中回放这些脚本。这使用户能够在开发软件应用程序的同时发现任何变更（无论是有意的变更还是缺陷）。

1. 配置环境

要开始针对应用程序记录脚本，必须先配置测试环境、配置测试应用程序和创建功能测试项目。

1）测试环境配置

测试环境配置如图 7.12 所示。

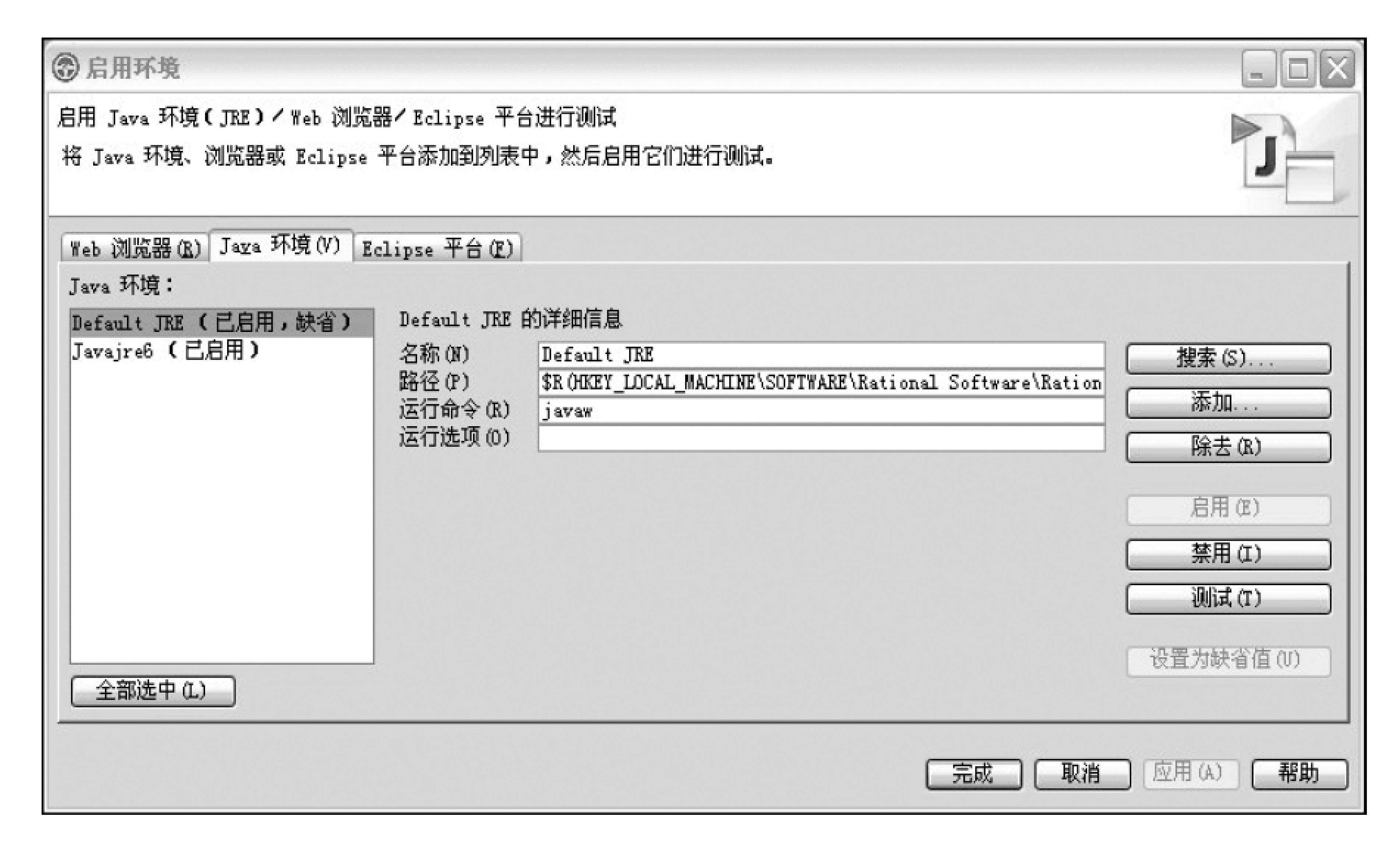


图 7.12 配置测试环境

选择“配置”→“启用环境进行测试”，将会看到“启用环境”窗口。窗口中有 3 个选项卡，分别是 Web 浏览器、Java 环境和 Eclipse 平台。环境配置的各项根据应用程序所采用的环

境来设定。

(1) 在“Web 浏览器”选项卡中，Internet Explorer 是测试回放的默认浏览器，并默认被启用。

(2) 在“Java 环境”选项卡中可以设计所使用的 JRE。软件默认的 JRE 是 IBM SDP Java Runtime Environment(JRE)。

(3) Eclipse 平台，Functional Tester 系统集成了 Eclipse，如果使用的是其他 Eclipse 或者 RCP，可以在这里设置。Functional Tester 支持 Eclipse 2.0 和 3.0、WebSphere 和 WorkBench 等平台。

2) 应用程序配置

应用程序配置如图 7.13 所示。



图 7.13 配置应用程序

这里可以选择要测试的应用程序，并且可以配置其参数，例如应用程序的参数以及所使用的 JRE 等，也可以运行应用程序。添加按钮，可以添加新的应用程序，也可以使用左侧更改测试程序。

2. 记录

当用户配置好应用程序和测试环境后就可以进行脚本记录了。下面以系统自带程序 ClassicsJavaA 为例进行脚本录制。

首先使用工具栏的“记录”按钮开始记录脚本。脚本中记录对应用程序的用户操作，例如按键和鼠标点击。还可插入验证点以测试应用程序中任何对象的数据或属性。记录过程中，验证点将捕获对象信息并将该信息存储在基线文件中；然后，在回放过程中，验证点将捕获对象信息并将该信息与基线进行比较。一旦停止记录，脚本和对象映射即写入项目目录。

1) 开始记录

要开始记录，首先创建 Functional Test 项目。项目名称为 Project1，如图 7.14 所示。

单击 Functional Test 工具栏中的“记录 Functional Test 脚本”按钮。

在脚本名称字段中输入 Classics(即将要使用的应用程序的名称)。

单击“完成”按钮，打开“记录监视器”窗口。

单击工具栏“插入脚本支持命令”按钮。这将打开“脚本支持功能”对话框，允许调用其他的脚本、在脚本中插入日志项、插入定时器、插入休眠命令(延时)或插入注释。

单击“关闭”按钮关闭“脚本支持功能”对话框。

图 7.14　创建项目

2）启动程序

选择来启动应用程序，接下来，脚本可以记录所有的鼠标事件和键盘事件。

要启动测试应用程序，单击工具栏“启动应用程序”按钮。

如有必要，在“启动应用程序”对话框中使用箭头来选择 ClassicsJavaA，并单击“确定”按钮。

3）记录操作

这里，测试人员可以对需要测试的应用程序做任何方式的操作。Functional Test 将会记录每一个操作过程。下面还是以 Functional Test 系统自带的范例程序 ClassicsJavaA 为例进行操作，具体操作过程如下所示。

（1）单击 Haydn 旁边的＋，展开 Composers 树中的文件夹。

（2）在列表中，单击 Symphonies Nos. 94 & 98。

（3）单击 Place Order 按钮。

（4）在 Member Logon 对话框中，保留 Existing Customer 和 Full Name 值为 Trent Culpito 的默认设置，如图 7.15 所示。不要在此时单击任何密码字段。

（5）单击 OK 按钮。

（6）在 card number 字段中输入信用卡号。在这里，必须使用 4 组四位数的有效格式，例如：7777 7777 7777 7777。

（7）在 expiration date 字段中输入采用有效格式的失效日期：12/14。

（8）单击 Place Order 按钮。

（9）单击订单确认消息框中的 OK 按钮，如图 7.16 所示。

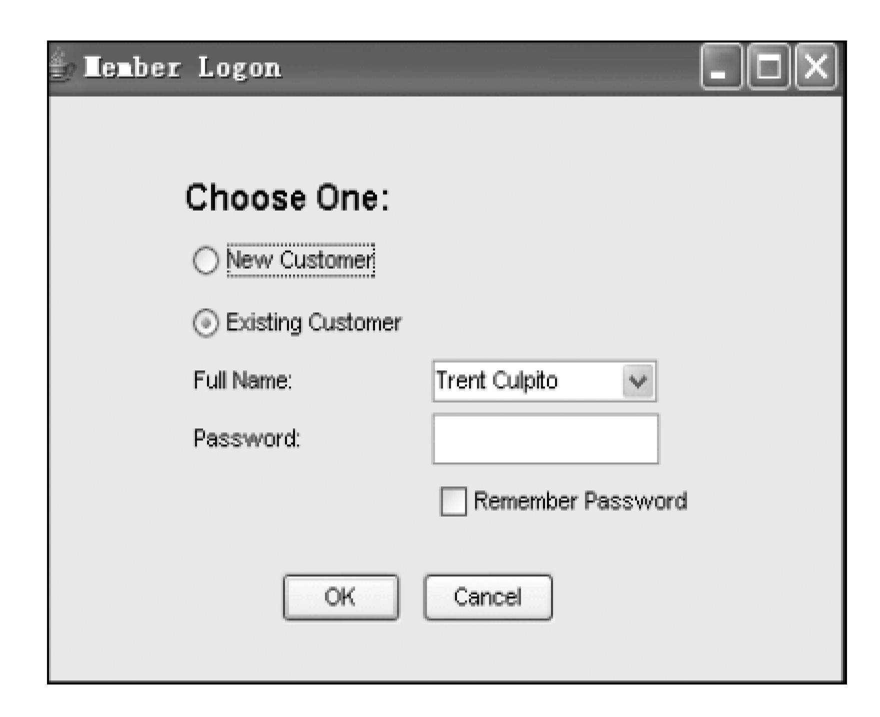


图 7.15　应用程序运行



图 7.16　应用程序运行完毕

(10) 关闭 ClassicsCD 应用程序。

(11) 在“记录监视器”窗口中,单击“停止记录”工具栏按钮。

观察脚本运行的过程。

运行结束后,Functional Test 会自动打开回放该脚本的日志内容,记录并分析该日志的内容。

3. 回放

脚本记录完成后,可以利用重复的测试方式对应用程序进行测试,如果系统环境更改了,也可以借助脚本回放功能测试已经记录下来的脚本。在 Functional Test 中脚本回放的操作如下所示。

(1) 单击 Functional Test 工具栏中的“运行 Functional Test 脚本”按钮。

(2) 观察脚本运行的过程。

(3) 运行结束后,Functional Test 会自动打开回放该脚本的日志内容,如图 7.17 所示。记录并分析该日志的内容。



图 7.17 回放记录

脚本回放是测试工具中很重要的一个环节,大部分测试工具都能提供脚本记录、回放等功能。重复性的测试可以借助脚本回放来实现,那么一般情况下,在什么场合会用到脚本回放功能呢?

(1) 测试开发阶段:使用用于记录的同一版本的被测试应用程序,回放脚本以验证脚本是否按预期工作。该阶段验证应用程序预期行为的基线。

(2) 回归测试阶段:回放脚本以将被测试应用程序的最新构建版本与测试开发阶段建立的基线进行比较。回归测试用于揭示自最后一次构建以来可能已引入应用程序的任何差异,可评估这些差异以确定它们是实际存在的缺陷还是故意的变更。

7.2.3 测试验证点的设置

除了脚本记录和回放功能，Functional Test 验证点跟踪也是经常会用到的功能之一。首先 Functional Test 会获取应用程序中不同的对象，针对不同的对象可以设置验证点，它可以验证某个操作是否已发生，也可以跟踪显示验证对象的状态，并比较其验证内容。

1. 获取应用程序中的对象

验证点的设置使用“正在记录”工具栏中的“插入验证点和动作命令”工具来完成。利用它来选择测试对象，如图 7.18 所示。

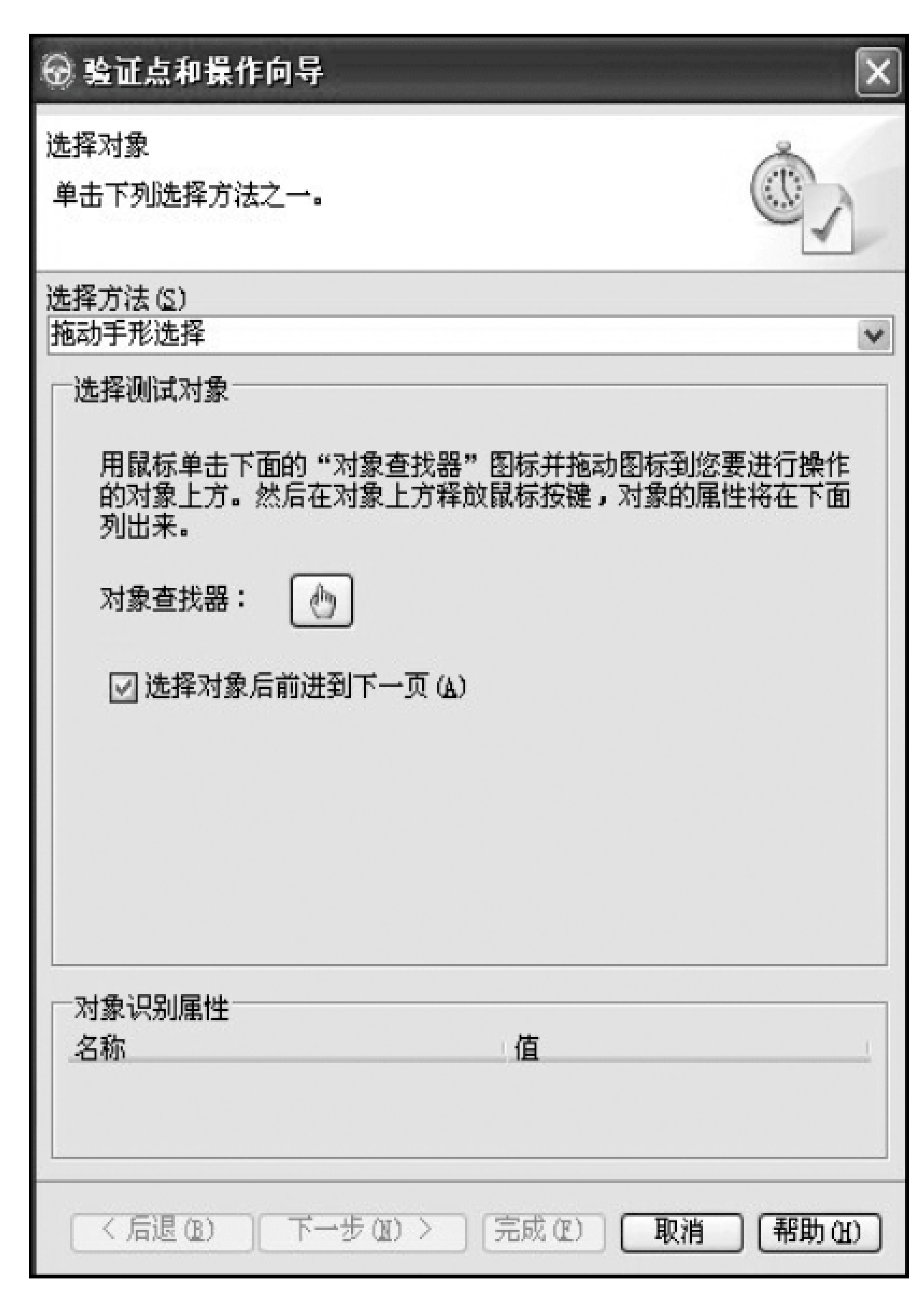


图 7.18 验证点设置界面

在使用的这个页面上选择需要执行测试的对象。当对象被选中时，其识别属性将显示在页面的底部。

如果“选择对象后前进到下一页”复选框被选中，那么当测试人员选择了错误的对象时，该对象的属性还是会被显示在底部，并继续后面的测试。一般情况下，可以取消“选择对象后前进到下一页”复选框。

在“验证点和操作向导”窗口中提供了三种选择对象的方法。

1）对象查找工具

这是选择一个对象最常见的和直接的方法。单击工具图标，鼠标光标变成工具形状，在应用程序中拖动对象，对象会突出显示，并被红色的框框住，如图 7.19 所示。当释放鼠标按钮时，选中对象的识别属性列在网格中。

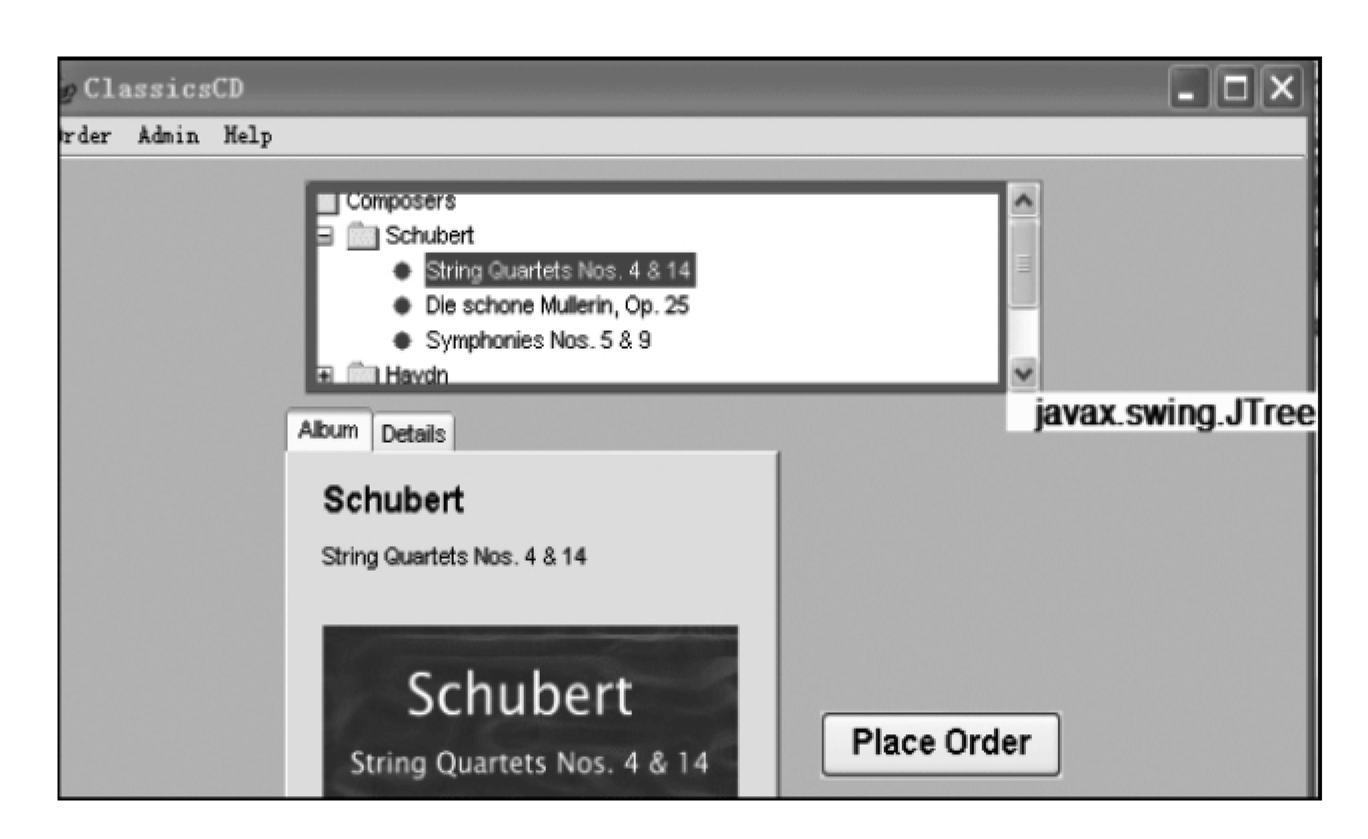


图 7.19 获取验证点

2）对象浏览器

使用这种方法，会显示一个对象的层次树，这些对象都是可测试的。在树的顶层显示已

经运行的应用程序。这是一个动态的视图,表示当前可用的对象。浏览对象树直到找到对象,然后单击选择该对象,之后它的识别属性将列在下面的网格中。

3) 延迟方法

这种方法使用 Object Finder 工具,但会产生延迟,由于延迟的设置,会给用户时间去选择一个需要的对象。用户首先点击其他对象,例如一个菜单命令。设置的秒数默认为 10,然后单击工具图标。移动鼠标悬停在应用程序,直到用户到达想要选择的对象。用户做的任何事在延迟周期都是没有记录的。这可以给用户足够的时间找到所需要的对象。例如,用户可能点击鼠标导致菜单弹出,选中菜单中某一项作为测试对象。

2. 设置验证点

在测试人员找到并记录需要插入验证点的对象后,接下来单击“验证点和操作向导”窗口中的“下一步”按钮,对找到的验证点进行设置。

Functional Test 可针对对象创建 7 种类型的验证点,即属性验证点和 6 种类型的数据验证点,分别为菜单层次结构、文本、表、树层次结构、列表和状态。

(1) 菜单层次结构:包括菜单层次结构和带有属性的菜单层次结构。

(2) 文本:包括可见文本。

(3) 表:包括表内容和选定表单元格。

(4) 树层次结构:包括树层次结构和选定树层次结构。

(5) 列表:包括列表元素。

(6) 状态:包括复选框状态和单选按钮状态。

使用属性验证点来测试应用程序内对象的属性。记录验证点时,将创建对象属性的基线。然后,当每次回放脚本时,将比较这些属性用来了解是否发生了任何变更。

数据验证点以同样的方式测试对象的不同数据类型。

可以对选中的对象执行以下操作(见图 7.20):

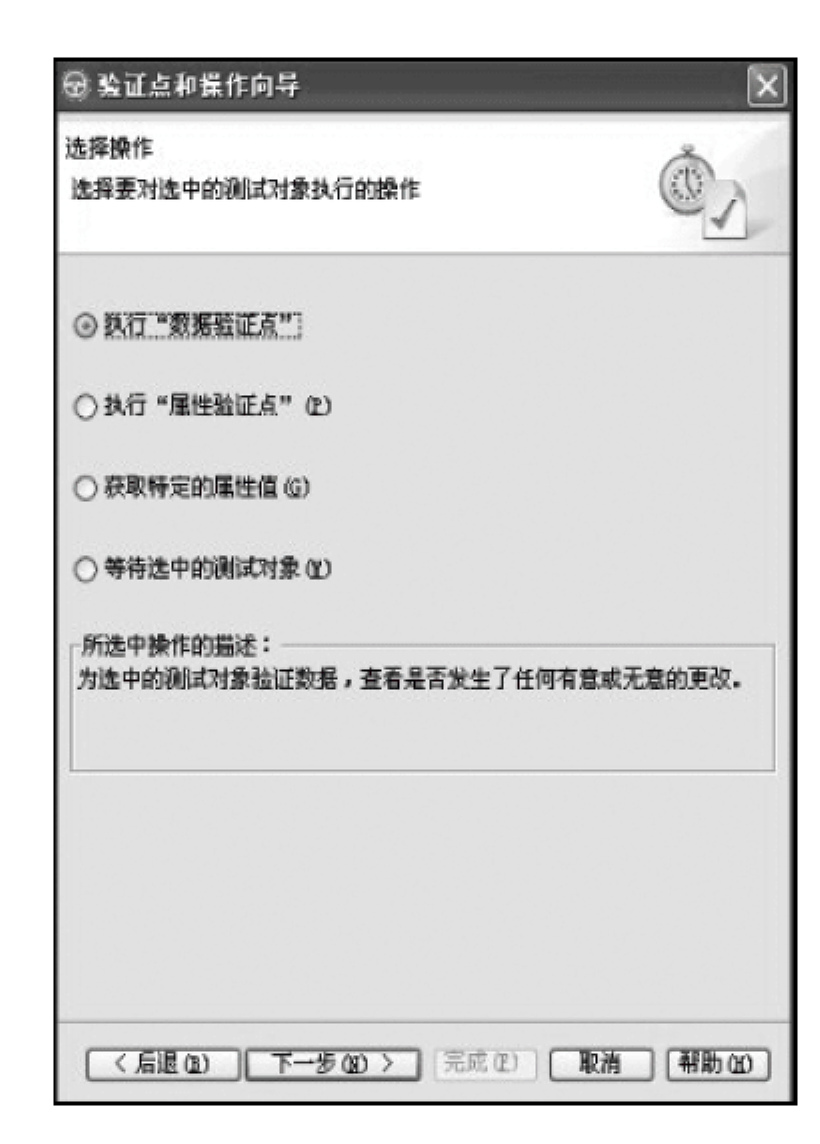


图 7.20 验证点操作向导 1

(1) 执行“数据验证点”:插入数据验证点命令,当回放记录脚本时,为选中的对象(例如 Jtree)记录数据信息。

(2) 执行“属性验证点”:插入属性验证点命令,当回放记录脚本时,为选中的对象(例如 Jlabel)记录所需的属性信息,并与其他脚本中该对象的属性值进行对比。

(3) 获取特定的属性值:向脚本中插入一个 getProperty 命令,在回放时会得到变量的属性值。

(4) 等待选中的测试对象:在脚本回放时,检测选中的测试对象是否存在,并等待其出现。

图 7.21 为所选中的需要插入验证点的对象。这里以树层次结构为例,Functional Test 可以针对此对象识别出树层次结构中所有的数据值,如图 7.22 所示。

可以按照以下操作步骤进行相关操作。



图 7.21　树层次结构



图 7.22　验证点操作向导 2

1）创建数据验证点

(1) 单击 Functional Test 工具栏中的“记录 Functional Test 脚本”按钮，打开“记录监视器”窗口。

(2) 运行应用程序。

(3) 在记录监视器中，单击“插入验证点或操作命令”按钮。

(4) 在“验证点和操作向导”的“选择对象”页面上，如果“选择对象后前进到下一页”复选框已选中，则将它清除。

(5) 使用对象查找器选择应用程序中的 Composers 树：单击对象查找器并将它拖动到树上。在按下鼠标左键时，将看到整个树绘有红色边框，并且对象名称(javax. swing. JTree)显示在红色边框旁边的屏幕提示中。当释放鼠标左键完成选择时，对象的识别属性显示在选择对象页面底部的网格中。

(6) 单击“下一步” 按钮。

(7) 在“选择操作”页面上，应该选中“执行‘数据验证点’”复选框，这是页面上的第一个操作，确保该项已被选中，并单击“下一步” 按钮。

(8) 在“插入验证点数据命令”页面上的“数据值”字段中选择“树层次结构”测试。该测试含有关于整个树形层次结构的信息。

(9) 在“验证点名称”字段中，输入 Classics_tree，并单击“下一步”按钮。

(10)“验证点数据”页面在右侧窗格的网格中显示已获取的数据。如果选中标记出现在某一项旁边的复选框中，则说明该项将被测试。默认情况下，所有项都被选中。使这些项保留选中状态。如果它们未被选中，则单击“全部选中”按钮。

(11) 单击“完成”按钮。

2）创建属性验证点

(1) 在 ClassicsCD 应用程序中，选择 Order→View Existing Order Status。请不要在此时单击任何密码字段。

(2) 单击 OK 按钮。将测试“查看现有订单”对话框中的标记“Trent Culpito 的订单”。

(3) 在记录监视器中,单击“插入验证点或操作命令”按钮。

(4) 这一次,在选择对象页面上选择“选择对象后前进到下一页”复选框(对象查找器下面的复选框)。

(5) 将对象查找器拖动到标记 Orderfor Trent Culpito 上以选中它。在按下鼠标左键时,将看到该标记被加上了红色边框,并且对象名称(javax. swing. JLabel)也显示出来。

(6) 选择“执行‘属性验证点’”,这是第二个操作。

(7) 单击“下一步”按钮。

(8) 在“插入属性验证点命令”页面上,保持“包括下级”字段设置为“无”。

(9) 在“验证点名称”下采用建议的默认值。

(10) 让“使用标准属性”选项保持选中状态,然后单击“下一步”按钮。

(11) 在“属性”列中,滚动到“文本属性”,选中该属性旁边的复选框,以在回放期间测试该属性。可能需要在复选框中单击两次才能使选中标记保留下来。

(12) 同时选中“不透明且可视属性”。

(13) 单击“完成”按钮。在 ClassicsCD 的 ViewExisting Orders 对话框中,单击 Close 按钮。

7.2.4 测试对象的映射

Functional Test 将被测应用程序中的各个元素识别为不同的对象,那么 Functional Test 对对象是如何进行管理的呢？这个可以通过“测试对象的映射”这一功能进行统一管理,而不需要在对象存在变更时逐一更改脚本。

在记录时,将对被测应用程序创建测试对象映射。对象映射提供了将对象添加到脚本的快捷方式。由于对象映射包含每个对象的识别属性,因此可在一个中心位置方便地更新信息。任何引用该对象映射的脚本也将共享已更新的信息。在映射文件中,对象与脚本之间的关系可以是专用的,也可以在脚本之间共享。

7.2.5 数据池的应用

数据池(datapool)是一个非常重要的,也是很常用的一项功能。数据驱动的命令向导和对不同类型数据表的支持功能,可以帮助测试人员便捷地完成测试任务。向导可以帮助测试人员在测试过程中选择需要输入数据的对象。当然,数据表中应记录所有的数据。

也就是说,在 Functional Test 中数据与测试脚本是分开的,这使得测试人员可以在不改变脚本的情况下,任意修改数据,添加新的测试用例。另外,测试数据还可以在多个脚本之间共享。

1. 数据驱动向导找到数据池应用的对象

(1) 录制脚本。

(2) 在“正在记录”工具栏中找到插入数据驱动的操作。

(3) 如图 7.23 所示,利用对象查找器进行对象的查找,这一操作和插入验证点时利用对象查找器的操作类似。

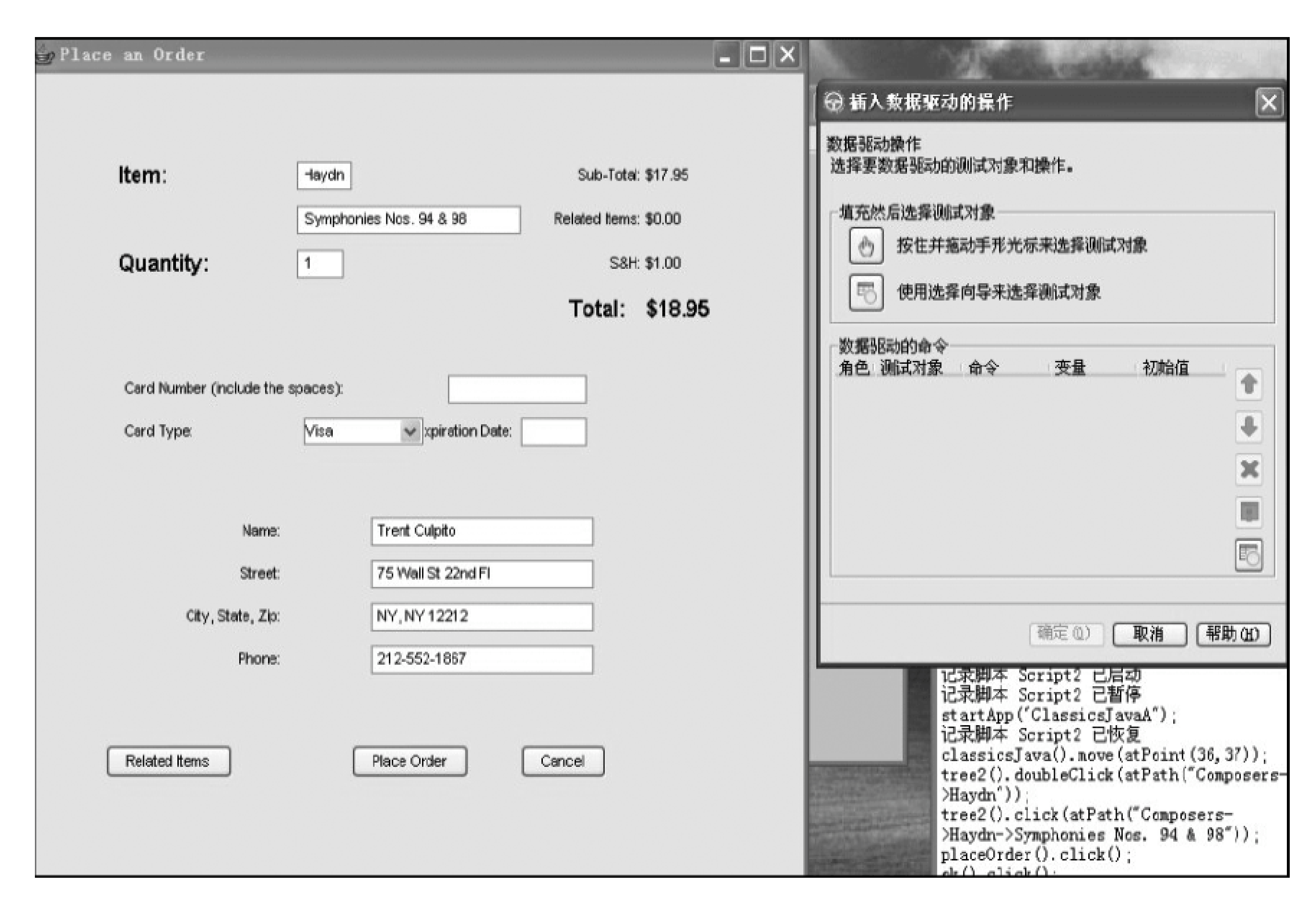


图 7.23　选择对象

图 7.24 显示出选中的对象。选中数据对象后，可以对其进行删除等操作。

2. 设计测试用例，生成数据池

数据池可以从外面数据导入，比如利用 Excel 生成数据，并保存为.cvs 格式，导入数据驱动中。也可以利用 Functional Test 来生成需要的数据，如图 7.25 所示。



图 7.24　选中对象

7.2.6　回归测试

回归测试一般发生在软件版本发生了变更的情况下，需要对原来所记录下来的测试脚本，令其在新版本的软件中重新运行，并完成测试。在 Functional Test 中是如何实现回归测试的呢？一般情况下需要设定应用程序，并更改脚本中的应用程序名。更改脚本中所选中的应用程序的操作为，打开脚本编辑器，改变 startApp 中应用程序的名称。

Functional Test 的测试是在对象的识别基础上进行的，也就是说，对象的定义对测试的过程影响是很大的。如果两个软件中某些对象发生了变化，那么“旧”版本基础上的脚本是否能在新版本中重新编译通过呢？答案当然是否定的。那么这时应该怎么办呢？这时候就需要更新对象图这个方法了。

在更新对象图中会列出 Functional Test 识别出的对象，如果在新版本中有的对象发生了变化，需要找到发生变化的对象，并替换“旧”版本相关的对象。

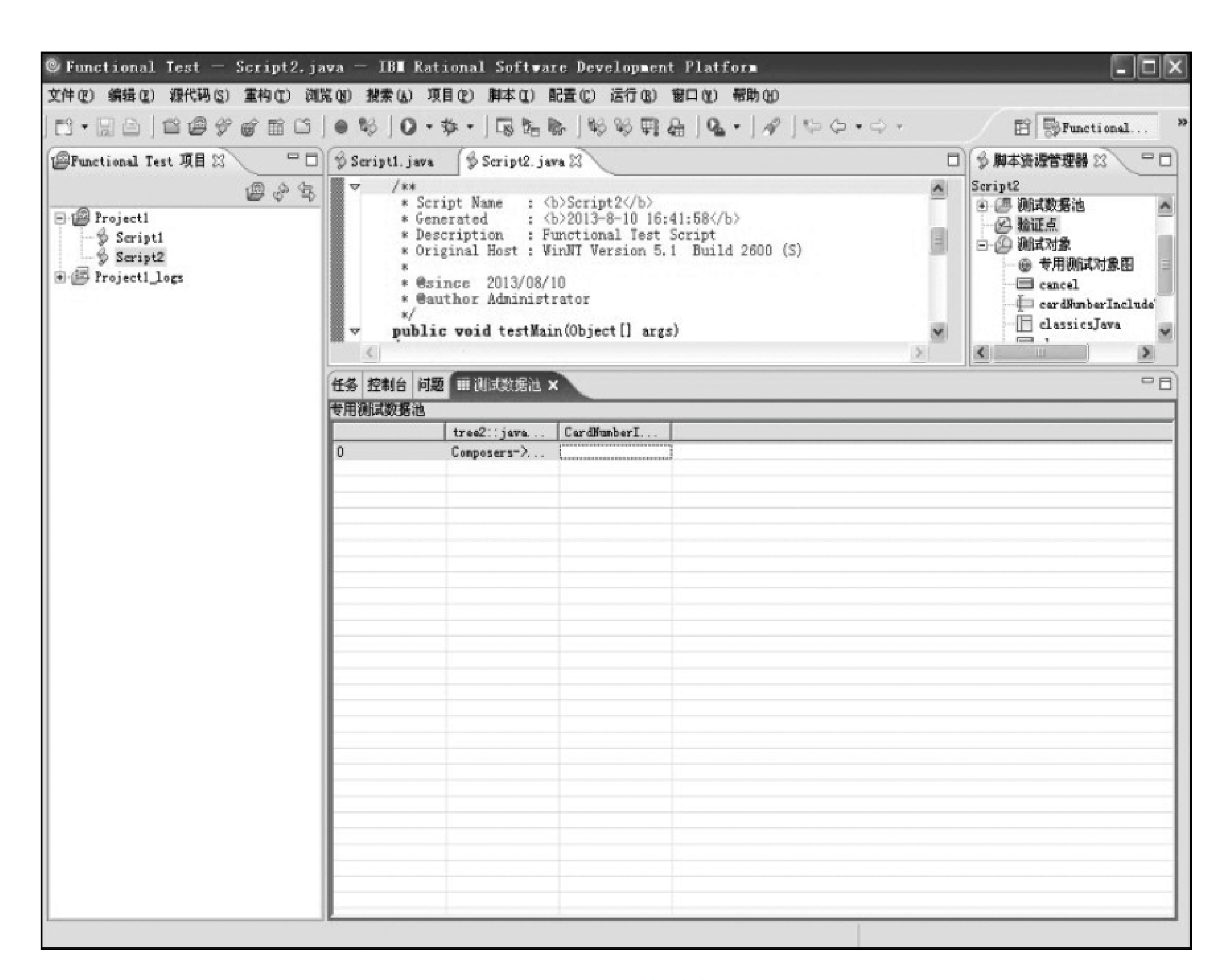


图 7.25 设计数据池

7.3 案例分析——图书管理系统软件测试

7.3.1 图书管理系统软件测试计划

本节简单分析一下图书管理系统的软件测试内容。一般情况下,应根据不同的软件架构进行不同的测试,本节是对 Functional Test 的应用,所以更多地考虑的是功能性的测试。而对于不同架构的软件,功能性的测试差别并不是很大。接下来就具体分析 C/S 以及 B/S 架构下软件测试的关注点差别。

采用 B/S 架构,客户端只能完成浏览、查询、数据输入等简单功能,而大部分工作由服务器承担,这对服务器造成了很大的压力。采用 C/S 结构时,客户端和服务器端都能够处理任务,这虽然对客户机的要求较高,但因此可以减轻服务器的压力。C/S 结构是建立在中间件产品基础之上的,要求应用开发者自己去处理事务管理、消息队列、数据的复制和同步、通信安全等系统级的问题。这对应用开发者提出了较高的要求,而且迫使应用开发者投入很多精力来解决应用程序以外的问题。

C/S 模式有以下特点:

(1) 由于客户端实现与服务器的直接相连,没有中间环节,因此响应速度快。

(2) 形式多样,可以充分满足客户自身的个性化要求。

(3) C/S 结构的管理信息系统具有较强的事务处理能力,能实现复杂的业务流程。

(4) 兼容性差,对于不同的开发工具,具有较大的局限性。若采用不同的工具,需要重新改写程序。

B/S 模式有以下特点:

(1) 具有分布性特点,可以随时随地进行查询、浏览等业务处理。

(2) 维护简单方便,只需要改变网页,即可实现所有用户的同步更新。

(3) 功能弱化,难以实现传统模式下的特殊功能要求。

在测试过程中,C/S模式主要集中到了多层结构(最常用的是三层的架构)、面向对象以及平台异构三个方面。所以针对C/S的测试主要是分析三个不同的层次。

(1) 客户端:这一层的测试,对于服务器和网络层是透明的,仅针对客户端进行测试。

(2) 客户端和服务器:客户端和服务器数据处理和交互的能力,例如数据吞吐量。但这一层不侧重于分析运行网络的测试。

(3) 体系结构:包括运行网络和性能测试。

C/S结构软件测试常用方法如下。

(1) 功能测试:客户端层次的测试常用的方法。应该被独立地执行,以揭示在其运行中的错误。

(2) 服务器测试:测试服务器的协调和数据管理功能,也考虑服务器性能(整体反应时间和数据吞吐量)。

(3) 数据库测试:测试服务器存储数据的精确性和完整性,检查客户端应用提交的事务,以保证其具备正确的存储、更新和检索。

(4) 事务测试:创建一系列的测试以保证每类事务都被按照要求处理。

(5) 网络通信测试:这些测试验证网络节点间的通信正常,消息传递无错地进行。

近年来,Web应用得到广泛的普及。一方面,在互联网浪潮的推动下,基于互联网的信息共享和电子商务不断发展,新浪、搜狐等大型网站不断涌现出来;另一方面,随着Java、CGI等网络技术的成熟,基于B/S结构的大型软件逐渐显示出巨大的优势。同时,什么样的服务器能够满足不同用户的需求,怎么能够保证Web服务器能够长期稳定地运行,为了解决这些问题,Web测试也就同样变得十分重要。

B/S结构软件在不同平台上都能使用,包括各种操作系统的应用、Web服务器、应用服务器、中间件、电子商务服务器、数据库服务器、防火墙以及浏览器等。Web应用系统具有多层体系结构,涉及客户端、服务器、数据通信和协议连接等。

B/S结构软件测试常用的类型如下。

(1) 基本功能测试:包括导航和连接测试、页面之间依赖关系测试、功能模块测试、数据流和信息流测试。

(2) 性能测试:包括负载测试和压力测试。

(3) 可用性测试:即GUI测试。

(4) 浏览器兼容性测试。

(5) 数据库测试:量级测试。

(6) 安全性测试:包括登录测试、信息是否进行加密以及未授权用户是否可以访问。

(7) 系统支持的协议的测试。

下面介绍对某图书管理系统进行测试的方法,这里主要是针对Functional Test而安排的测试,所以主要考虑了功能性测试的部分。

测试计划的部分内容如表7.3所示。

这里主要针对图书管理系统的功能测试进行分析,如表7.4所示。

表 7.3 测试计划部分内容

测 试 计 划			
测试计划概述			
项目名称	图书管理系统	测试开始日期	2013.1
测试产品版本	1.0	提交版本日期	
开发人员		测试人员	
产品经理		发布日期	
测试阶段	测试周期		任务安排
第一阶段	数据和数据库的完整性测试		
第二阶段	接口测试		
第三阶段	集成测试		
第四阶段	功能测试		
第五阶段	GUI 测试		
第六阶段	性能测试		
参考文档			
测试环境			

表 7.4 图书管理系统的功能测试分析

测试需求描述	测试项描述	期望结果	优先级别
登录功能测试	合理性	有提示	1
	合法性	有提示	1
	系统操作成功	进入系统	2
	系统操作失败	有提示	2
借书测试	图书书号提交	图书显示	2
	图书缺失	不能借阅	2
	借阅过程成功	成功操作	1
	借阅过程失败	有提示	1
还书测试	合理性检查	有提示	2
	还书成功	成功操作	1
	还书失败	有提示	1
注册测试	合理合法性	有提示	1
	添加用户	成功操作	1
	修改用户信息，删除用户	成功操作	1
图书管理测试	图书查询(按书号)	成功操作	2
	图书查询(按书目名称)	成功操作	1
	增加图书	成功操作	1
	编辑、删除图书	成功操作	1

7.3.2 图书管理系统黑盒测试用例设计

这里以“借书模块”为例，进行黑盒测试用例设计，如表7.5所示。

表7.5 “借书模块”黑盒测试用例设计表

测试项描述	期望结果	优先级别
图书书号提交	图书显示	2
图书缺失	不能借阅	2
借阅过程成功	成功操作	1
借阅过程失败	有提示	1

在借书过程中，需要进行以下几个方面的测试：

(1) 能够进行图书的查询，这个查询只需要按书号查询即可。

(2) 如果图书缺失，不能提供借阅。

(3) 如果图书存在，可以提供借阅，并完成借阅操作。

(4) 借阅过程不成功。

由于该图书管理系统是按照面向对象的思想设计的，因此，考虑采用事件流的方法来进行测试用例分析，如图7.26所示。当然具体到书目查询等内容，可以考虑采用等价类或者其他方法来进行用例设计。

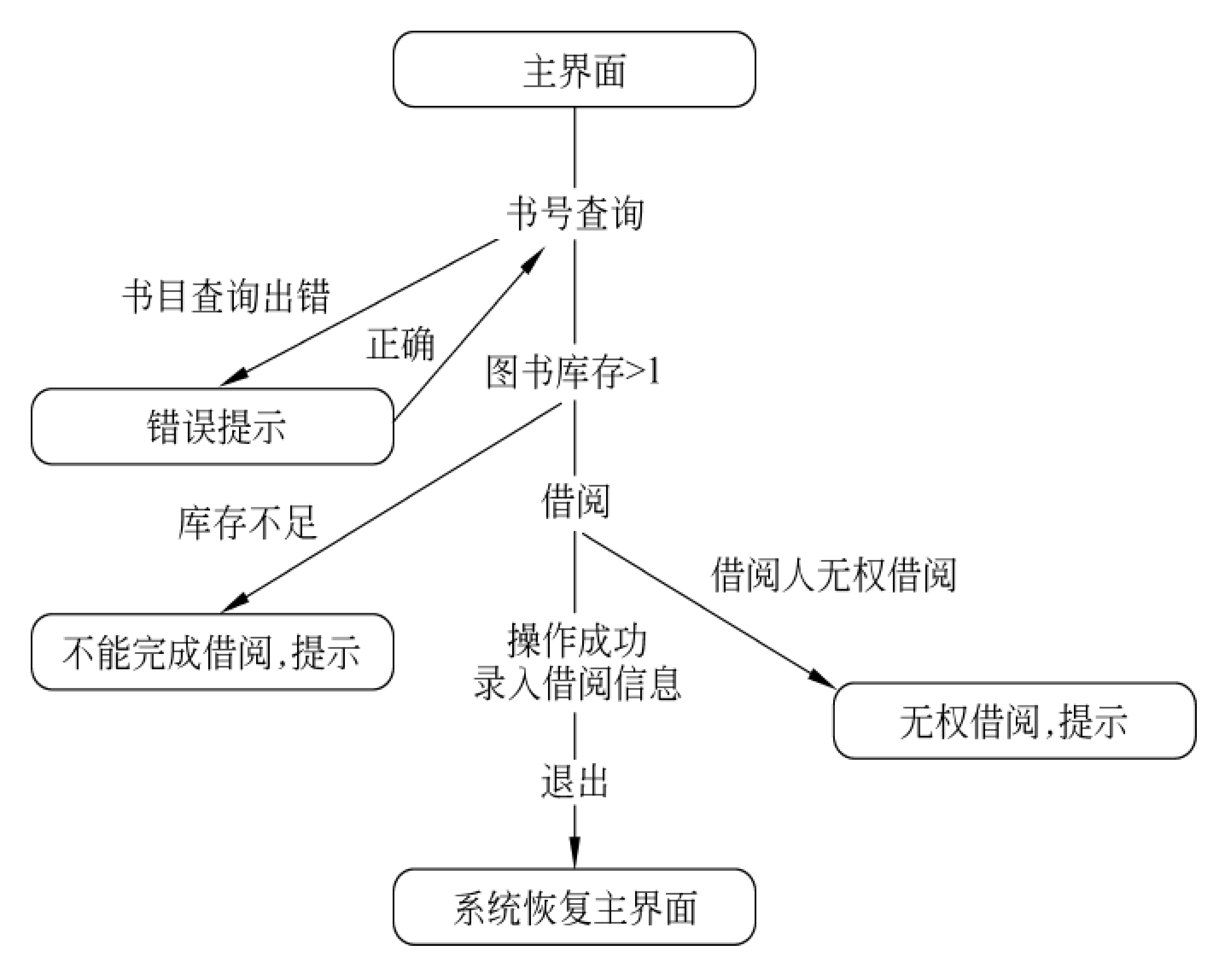


图7.26 场景法分析测试用例

基本流：当借阅者从图书馆借阅某书刊时，用例启动，通过书目查询找到需借阅的图书，进行借阅，创建借阅记录，录入借阅人、借阅时间和归还日期等信息，完成借阅。

分支流：有以下3个，分别是书目查询出错、库存不足以及借阅人无权借阅。

7.3.3 利用 Functional Test 测试

1. 配置应用程序环境

首先需要配置应用程序的环境，如图7.27所示。根据需要定义应用程序的名称。这里

应用程序可以是 Java 或者 HTML 等不同的形式，如图 7.28 所示。



图 7.27　配置应用程序环境



图 7.28　选择 HTML 应用程序

2. 记录脚本

这里以 HTML 格式为例，记录应用程序运行脚本，如图 7.29 所示。

```
import resources.Script3Helper;

/**
 * Description    : Functional Test Script
 * @author Administrator
 */
public class Script3 extends Script3Helper
{
    /**
     * Script Name    : <b>Script3</b>
     * Generated      : <b>2005-10-13 0:10:26</b>
     * Description    : Functional Test Script
     * Original Host : WinNT Version 5.1  Build 2600 (S)
     *
     * @since  2005/10/13
     * @author Administrator
     */
    public void testMain(Object[] args)
    {
        startApp("图书馆里系统");

        // HTML Browser
        browser_htmlBrowser(document_河北科技大学图书馆v50书目检索系统(),DEFAULT_FLAGS).inputKeys("a{BKSP}rua{BKSP}{BKS
        browser_htmlBrowser(document_河北科技大学图书馆v50书目检索系统(),MAY_EXIT).inputChars("软件测试");
    }
}
```

图 7.29　脚本记录

上面介绍的用例是系统自带的,用户也可以使用自己的程序进行测试,测试方法与上面介绍的步骤类似。

7.4 本章小结

本章介绍了黑盒测试工具——IBM Rational Function Tester。首先介绍了 Function Test 的操作平台以及 Function Test 安装过程,其次详细介绍了 Function Test 的操作流程、环境配置和脚本记录,最后针对 Function Test 提供的数据驱动测试进行了单独的分析介绍,其中包括数据验证点的设置和数据池的操作。

习 题

1. 黑盒测试工具可以分成哪几类?简述常用的几种黑盒测试工具。
2. Functional Tester 提供了哪些插入验证点的方法。
3. 什么是回归测试?利用 Functional Tester 如何进行回归测试?
4. Functional Tester 是否提供数据驱动测试模式?如何使用数据池?

第8章　白盒测试法案例分析

软件测试总体来说分为白盒测试和黑盒测试，这两类测试下又有不同的测试工具。通过这些工具，使得软件中的问题直观的显示出来，帮助测试人员更好的找出软件的错误。本章主要介绍白盒测试的相关工具：JUnit 框架测试及 HtmlUnit 测试。通过本章的学习，读者应该对白盒测试的相关工具有一个大致的了解，并熟悉 JUnit 和 HtmlUnit 测试框架，掌握 JUnit 测试的方法。

8.1　白盒测试工具介绍

白盒测试工具一般针对代码进行测试，测试中发现的缺陷可以定位到代码级。由于白盒测试工具多用于单元测试阶段，因此也被称为单元测试工具。单元测试不仅要验证被测单元的功能实现是否正确，还要查找代码中的内存使用错误和性能瓶颈，并且为了检验测试的全面性，还要对测试所达到的覆盖率进行统计和分析。因此，白盒测试工具多为一个套件，其中包含了动态错误检测、时间性能分析和覆盖率统计等多个工具。根据测试原理不同，可分为静态测试工具和动态测试工具。

8.1.1　静态测试工具

静态测试工具不需要运行代码，只需要对代码进行静态分析即可。静态测试工具主要是对代码进行语法扫描，发现编码中不符合规范的地方。静态测试工具的主要代表有 TeleLogic 公司的 Logiscope、Macabe 公司的 Macabe 和 PR 公司的 PRQA 等。

其中 TeleLogic 公司的 Logiscope 实际上是一组工具集，包括代码质量度量（Audit）、代码规则检查（RuleChecker）、高可靠性规则集（MISRA Rule）和边缘覆盖率统计（TestChecker）4 个模块。Logiscope 贯穿于软件开发的各个阶段，是面向源代码进行工作的。

Logiscope 支持多平台及多语言环境，比如可工作在 Windows 环境下，也可工作在 UNIX 环境下，可支持 C、C++ 和 Java 等多种编程语言。

Logiscope 的特点如下：

（1）在软件开发过程中的各个阶段，Logiscope 的使用能够帮助软件工程师开发出最优秀的代码，使代码得到充分测试，降低维护工作量。

（2）Logiscope 能够有效地进行代码走查，可自动检测软件编码。

（3）在复查阶段，Logiscope 可定位出错较多的模块，以便有针对性地对代码复查。

（4）Logiscope 可标识出未彻底测试的代码段，以排除代码中隐藏的错误。

（5）项目管理人员利用 Logiscope 可有效地管理和监督整个软件开发过程。

（6）Logiscope 对测试的结果可以 Word 或 HTML 形式进行显示，结果一目了然。

8.1.2 动态测试工具

动态测试工具区别于静态测试工具的主要地方是需要实际运行软件系统。通常，这类测试工具一般会采用“插桩”的方式，即向代码生成的可执行文件中插入一些检测代码，用于统计程序运行时的数据。动态测试工具的主要代表有 Compuware 公司的 DevPartner、Rational 公司的 Purify 系列和 Numega 公司的 BounceChecker 等。

其中，Purify 是面向 VC、VB 或者 Java 开发的，能够测试 Visual C/C++ 和 Java 代码中与内存有关的错误，确保整个应用程序的质量和可靠性。在查找典型的 Visual C/C++ 程序中的传统内存访问错误以及 Java 代码中与垃圾内存收集相关的错误方面，Rational Purify 可以大显身手。Rational Robot 的回归测试与 Rational Purify 结合使用能够完成可靠性测试。

8.2 JUnit 框架测试

8.2.1 JUnit 框架介绍

JUnit 是为 Java 程序开发者实现单元测试提供的一种框架。它由 Kent Beck 和 Erich Gamma 建立。目前，多数 Java 的开发环境都已经集成了 JUnit 作为单元测试的工具，如 NetBeans、Eclipse 和 MyEclipse 等。使得 Java 单元测试更规范有效，并且更有利于测试的集成。JUnit 的一大主要特点是，它在执行的时候，各个方法之间是相互独立的，一个方法的失败不会导致别的方法失败，方法之间也不存在相互依赖的关系，彼此是独立的。目前，主要使用 JUnit3 和 JUnit4 测试框架。进行测试时，JUnit3 主要是通过继承 TestCase 类来撰写测试用例，使用 testXXX()名称来撰写单元测试。使用 JUnit3 进行测试的过程如下：

（1）一个 import 语句引入所有 junit.framework.* 下的类。

（2）一个 extends 语句让测试类从 TestCase 继承。

（3）一个调用 super(string)的构造函数。

如使用 JUnit3 框架的测试类 MyMathTest.java 的形式如下：

```
package calculator;
import junit.framework.TestCase;              //import 语句引入 junit.framework.TestCase 类
public class MyMathTest extends TestCase {    //继承 TestCase 类
    public MyMathTest(String testName) {
        super(testName);                      //调用 super(string)构造函数
    }
}
……
```

在 JUnit4 框架中，不再需要继承 TestCase 类。在 JUnit3 中需要覆盖 TestCase 类中的 setUp 和 tearDown 方法，其中 setUp 方法会在测试执行前被调用以完成初始化工作，而 tearDown 方法则在结束测试结果时被调用，用于释放测试中使用的资源。而在 JUnit4 中，只需要在方法前加上@Before 或@BeforeClass 标识初始化工作，使用@After 或@AfterClass 标识收尾工作；在 JUnit4 中使用@Test 标识代表这个方法是测试用例中的测

试方法。在写程序前做了很好的规划,那么哪些方法是什么功能都应该事先确定下来。因此,即使该方法尚未完成,它的具体功能也是确定的,这也就意味着可以为它编写测试用例。但是,如果已经把该方法的测试用例写完,但该方法尚未完成,那么测试的时候一定是“失败”。这种失败和真正的失败是有区别的,因此 JUnit 提供了一种方法来区别它们,那就是在这种测试函数的前面加上@Ignore 标识,这个标识的含义就是“某些方法尚未完成,暂不参与此次测试”。这样,测试结果就会提示用户有几个测试被忽略,而不是失败。一旦完成了相应函数,只需要把@Ignore 标记删去,就可以进行正常的测试。

如使用 JUnit4 框架的测试类 MyMathTest.java 的形式如下:

```
package calculator;
import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;      //提供 Assert 方法
public class MyMathTest {
    public MyMathTest() {
    }
    @BeforeClass         //表明是初始化方法,只能有一个,而且在所有测试方法运行前只运行一次
    public static void setUpClass() throws Exception {
    }
    @AfterClass                    //所有测试执行完毕之后进行收尾工作,只能有一个
    public static void tearDownClass() throws Exception {
    }
    @Ignore("Multiply() Not yet implemented")      //未完成,暂不参与测试
    @Test                          //表明下面的方法是测试方法
……
```

在执行测试时,有 Failure 与 Error 两种测试尚未通过的信息。

Failure 是预期的结果与实际运行单元的结果不同所导致的,例如,当使用 assertEquals()或其他 assertXXX()方法断言失败时,就会回报 Failure,这时候要检查单元方法中的逻辑设计是否有误。

Error 指的是程序没有考虑到的情况,在断言之前程序就因为某种错误引发例外而终止。例如,在单元中存取某个数组,因为存取超出索引而引发 ArrayIndexOutOfBounds-Exception,这会使得单元方法无法正确完成,在测试运行到 assertXXX()前就提前结束,这时候要检查单元方法中是否有未考虑到的情况而引发流程突然中断。

1. JUnit 的各种断言

JUnit 提供了一些辅助函数,用于帮助确定某个被测函数是否工作正常,通常把所有这些函数统称为断言。断言是单元测试最基本的组成部分。

(1) assertEquals([String message], expected,actual)

判断期待结果和实际结果是否相等,expected 填写期待结果,actual 填写实际结果,也就是通过计算得到的结果。写好之后,JUnit 会自动进行测试并把测试结果反馈给用户。

(2) assertTrue([String message],boolean condition)

对布尔值求值,看它是否为"真"。

(3) assertFalse([String message],boolean condition)

对布尔值求值,看它是否为"假"。

(4) assertNull([String message],java.lang.Object object)

检查对象是否为"空"。

(5) assertNotNull([String message],java.lang.Object object)

检查对象是否不为"空"。

(6) assertSame([String message],expected,actual)

检查两个对象是否为同一实例。

(7) assertNotSame([String message],expected,actual)

检查两个对象是否不为同一实例。

(8) fail(String message)

使测试立即失败,其中 message 参数是可选的。这种断言通常被用于标记某个不应该到达的分支(例如在一个预期发生的异常之后)。

2. JUnit 中常用的接口和类

1) Test 接口

Test 接口主要用于运行测试和收集测试结果,它使用了 Composite 设计模式,是单独测试用例(TestCase)、聚合测试模式(TestSuite)及测试扩展(TestDecorator)的共同接口。

它的 public int countTestCases()方法用来统计这次测试有多少个 TestCase,另外一个方法就是 public void run(TestResult),TestResult 是用例接受测试的结果,run 方法执行本次测试。

2) TestCase 抽象类

TestCase 抽象类是定义测试中固定的方法。TestCase 是 Test 接口的抽象实现(不能被实例化,只能被继承),其构造函数 TestCase(string name)根据输入的测试名称 name 创建一个测试实例。由于每一个 TestCase 在创建时都要有一个名称,若某测试失败了,便可识别出是哪个测试失败。

TestCase 类中包含 setUp()和 tearDown()方法。setUp()方法集中初始化测试所需的所有变量和实例,并且在依次调用测试类中的每个测试方法之前再次执行 setUp()方法。tearDown()方法则是在每个测试方法之后释放测试程序方法中引用的变量和实例。

实际编写测试用例时,只需继承 TestCase,完成 run 方法即可,然后 JUnit 获得测试用例,执行它的 run 方法,把测试结果记录在 TestResult 之中。

3) Assert 静态类

Assert 静态类是一系列断言方法的集合。Assert 包含了一组静态的测试方法,用于期望值和实际值比对以判断是否正确。如果测试失败,Assert 类就会抛出一个 AssertionFailedError 异常,JUnit 测试框架将这种错误归入 Failes 并加以记录,同时标志为未通过测试。如果该类方法中指定一个 String 类型的传递参数,则该参数将被作为 AssertionFailedError 异常的标识信息,告诉测试人员该异常的详细信息。

JUnit 提供了很多断言方法,包括基础断言、数字断言、字符断言、布尔断言和对象断

言等。

其中 assertEquals(Object expected,Object actual)内部逻辑判断使用 equals()方法,这表明断言两个实例的内部哈希值是否相等时,最好使用该方法对相应类实例的值进行比较。而 assertSame(Object expected,Object actual)内部逻辑判断使用了 Java 运算符"==",这表明该断言判断两个实例是否来自同一个引用(Reference),最好使用该方法对不同类的实例的值进行比对。assertEquals(String message,String expected,String actual)方法对两个字符串进行逻辑比对,如果不匹配则显示两个字符串有差异的地方。ComparisonFailure 类提供两个字符串的比对,不匹配则给出详细的差异字符。

4) TestSuite 测试包类

TestSuite 是多个测试的组合。TestSuite 类负责组装多个 TestCase。待测类中可能包括了对被测类的多个测试,而 TestSuite 负责收集这些测试,使测试人员可以在一个测试中完成全部的对被测类的多个测试。

TestSuite 类实现了 Test 接口,且可以包含其他的 TestSuite。它可以处理加入 Test 时所有抛出的异常。

TestSuite 处理测试用例有以下 6 个规约(否则会被拒绝执行测试):

(1) 测试用例必须是公有类(Public)。

(2) 测试用例必须继承自 TestCase 类。

(3) 测试用例的测试方法必须是公有的(Public)。

(4) 测试用例的测试方法必须被声明为 void。

(5) 测试用例中测试方法的前置名词必须是 test。

(6) 测试用例中测试方法无任何传递参数。

5) TestResult 结果类和其他类与接口

TestResult 结果类集合了任意测试的累加结果,通过 TestResult 实例传递给每个测试的 Run()方法。TestResult 在执行 TestCase 时如果失败会抛出异常。

TestListener 接口是一个事件监听规约,可供 TestRunner 类使用。它通知 listener 对象相关事件,方法包括测试开始 startTest(Test test)、测试结束 endTest(Test test)、增加错误 addError(Test test,Throwable t)和增加失败 addFailure(Test test,AssertionFailedError t)。

TestFailure 失败类是"失败"状况的收集类,解释每次测试执行过程中出现的异常情况。其 toString()方法返回"失败"状况的简要描述。

8.2.2 案例分析——利用 JUnit 测试计算器程序

这里的计算器程序是在 NetBeans 中实现的,在 NetBeans 中的 JUnit 测试方法如下。

第一步:在 NetBeans 中新建一个项目,并新建包 calculator,在该包中建立若干类,实现一个类似于 Windows 计算器的简单程序,主要用于测试加、减、乘、除运算的功能。其中实现四则运算的类为 MyMath.java,其代码如下:

```
package calculator;
import java.math.BigDecimal;
```

```
public class MyMath {
private static int DEFAULT_SCALE=20;
    private static BigDecimal first;
    private static BigDecimal second;
    private static BigDecimal getBigDecimal(double number) {
                                                    //将 double 类型转换成 BigDecimal 类型
        return new BigDecimal(number);
    }
    public static double add(double num1, double num2) {            //加
        //调用工具方法将 double 转换成 BigDecimal
        first =getBigDecimal(num1);
        second =getBigDecimal(num2);
        return first.add(second).doubleValue();
    }
    public static double subtract(double num1, double num2) {       //减
        first =getBigDecimal(num1);
        second =getBigDecimal(num2);
        return first.subtract(second).doubleValue();
    }
    public static double multiply(double num1, double num2) {       //乘
        first =getBigDecimal(num1);
        second =getBigDecimal(num2);
        return first.multiply(second).doubleValue();
    }
    public static double div(double num1, double num2) {            //除
        first =getBigDecimal(num1);
        second =getBigDecimal(num2);
        return first.divide(second, DEFAULT_SCALE, BigDecimal.ROUND_HALF_UP).
        doubleValue();
    }
}
```

第二步：检查项目下的“测试库”中是否有 JUnit3、JUnit4 的单元测试包，如果没有，通过在“测试库”上右击，选择“添加库”命令添加单元测试包。

第三步：生成 JUnit 测试框架。在要测试的 MyMath.java 类上右击，在弹出的快捷菜单中依次选择“工具”→“创建 JUnit 测试”，或按下 Ctrl+Shift+U 键，如图 8.1 所示。

第四步：在弹出的“选择 JUnit 版本”对话框中选择创建测试框架的 JUnit 版本，有 JUnit 3.x 和 JUnit 4.x 两个版本，如图 8.2 所示。选择其中一个并单击“选择”按钮。

第五步：在弹出的如图 8.3 所示的“创建测试”对话框中选择测试类的位置，默认是在项目中的“测试包”中，并选择“代码生成”的相关选项，单击“确定”按钮，在项目的“测试包”中新增 calculator 项目，并在项目下新增 MyMath.java 类的测试类 MyMathTest.java 类，同时在编辑窗口显示自动生成的 MyMathTest.java 测试类的源代码。

JUnit 3.x 生成的测试类代码如下：



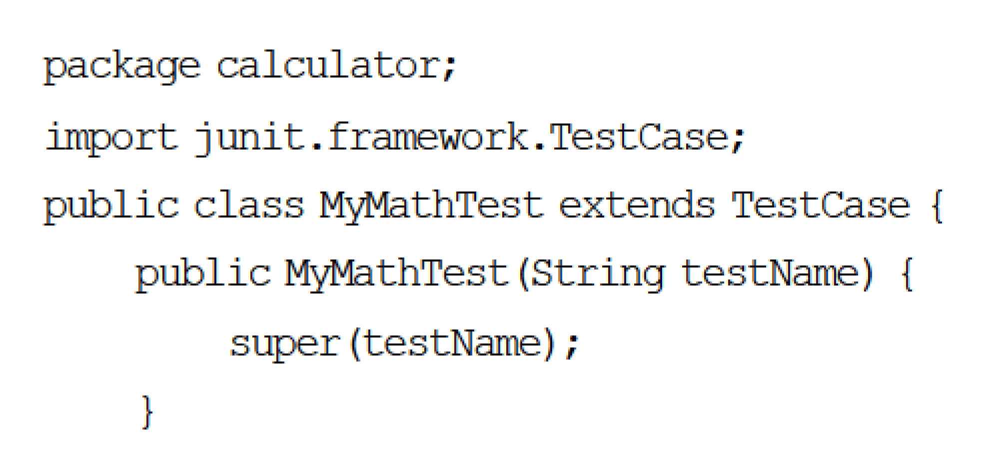
图 8.1　创建 JUnit 测试的菜单选项



图 8.2　“选择 JUnit 版本”对话框



图 8.3　“创建测试”对话框



```
package calculator;
import junit.framework.TestCase;
public class MyMathTest extends TestCase {
    public MyMathTest(String testName) {
        super(testName);
    }
```

```
/**
 * Test of add method, of class MyMath.
 */
public void testAdd() {
    System.out.println("add");
    double num1 = 0.0;
    double num2 = 0.0;
    double expResult = 0.0;
    double result = MyMath.add(num1, num2);
    assertEquals(expResult, result, 0.0);
    //TODO review the generated test code and remove the default call to fail.
    //fail("The test case is a prototype.");
}

/**
 * Test of subtract method, of class MyMath.
 */
public void testSubtract() {
    System.out.println("subtract");
    double num1 = 0.0;
    double num2 = 0.0;
    double expResult = 0.0;
    double result = MyMath.subtract(num1, num2);
    assertEquals(expResult, result, 0.0);
    //TODO review the generated test code and remove the default call to fail.
    //fail("The test case is a prototype.");
}

/**
 * Test of multiply method, of class MyMath.
 */
public void testMultiply() {
    System.out.println("multiply");
    double num1 = 0.0;
    double num2 = 0.0;
    double expResult = 0.0;
    double result = MyMath.multiply(num1, num2);
    assertEquals(expResult, result, 0.0);
    //TODO review the generated test code and remove the default call to fail.
    //fail("The test case is a prototype.");
}

/**
 * Test of div method, of class MyMath.
 */
public void testDiv() {
    System.out.println("div");
```

```
        double num1 = 0.0;
        double num2 = 0.0;
        double expResult = 0.0;
        double result = MyMath.div(num1, num2);
        assertEquals(expResult, result, 0.0);
        //TODO review the generated test code and remove the default call to fail.
        //fail("The test case is a prototype.");
    }

}
```

JUnit 4.x 生成的测试类代码如下：

```
package calculator;
import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;
public class MyMathTest {
    public MyMathTest() {
    }

    @BeforeClass
    public static void setUpClass() throws Exception {}

    @AfterClass
    public static void tearDownClass() throws Exception {}
    /**
     * Test of add method, of class MyMath.
     */
    @Test
    public void testAdd() {
        System.out.println("add");
        double num1 = 0.0;
        double num2 = 0.0;
        double expResult = 0.0;
        double result = MyMath.add(num1, num2);
        assertEquals(expResult, result, 0.0);
        //TODO review the generated test code and remove the default call to fail.
        //fail("The test case is a prototype.");
    }
    /**
     * Test of subtract method, of class MyMath.
     */
    @Test
    public void testSubtract() {        //此处代码和 JUnit 3.x 生成的代码相同
```

```
    }
    /**
     * Test of multiply method, of class MyMath.
     */
    @Test
    public void testMultiply() {        //和 JUnit 3.x生成的代码相同
    }
    /**
     * Test of div method, of class MyMath.
     */
    @Test
    public void testDiv() {             //和 JUnit 3.x生成的代码相同
    }
}
```

第六步：执行测试。依次选择菜单命令“运行”→“测试项目”，在窗口下方的控制台中出现测试结果，如图 8.4 所示。进度条为绿色表示通过的测试，红色表示未通过的测试。该测试中有 3 个测试通过，一个测试出现错误，主要是因为测试中进行了“除 0”的操作。

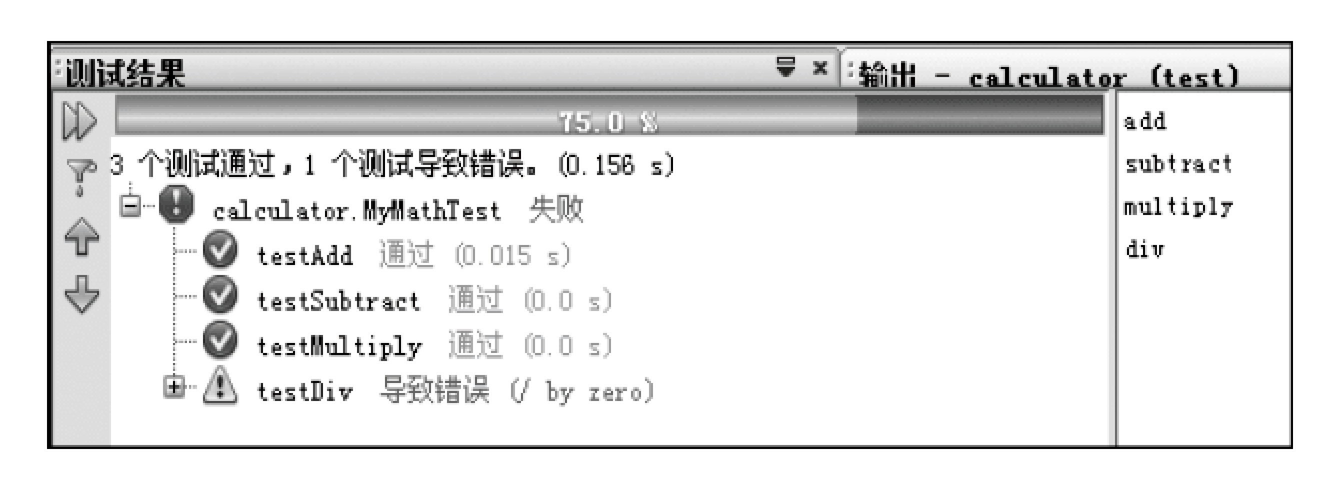


图 8.4　JUnit 测试结果

通过以上的例子可以了解 JUnit 测试的基本过程、测试类的基本框架结构及测试类中的相关符号表示的含义。在所生成的测试类的基础上可以进行修改，实现高级测试，下面介绍 JUnit 的高级应用。

8.3　JUnit 的高级应用

8.3.1　限时测试

在程序设计过程中，可能会出现一些逻辑结构比较复杂的程序，这些复杂结构可能会导致程序运行过程中出现死循环。为了解决这个问题，JUnit 提供了相应的措施，即限时测试。在测试中可对@Test 加上对应的参数，如下面的程序片段利用 timeout 设定时间，单位为毫秒。

```
@Test(timeout = 1000)
    public void squareRoot(){
        calculator.squareRoot(9);
        assertEquals(3, calculator.getResult());
}
```

8.3.2 测试异常

在程序设计中，异常处理也是非常重要的，因此，在程序中往往会抛出异常。程序运行中，如果该抛出异常但是却没有抛出，这种情况也可以看作是测试中的一种缺陷。JUnit 也提供了测试异常是否能够正常抛出的方法，可在@Test 后添加相应的参数，如下面的程序片段，可利用 expected 属性来测试是否抛出了指定的异常。

```
@Test(expected =ArithmeticException.class)
    public void testDiv() {
      System.out.println("div");
      double num1 =0.0;
      double num2 =0.0;
      double expResult =0.0;
      double result =MyMath.div(num1, num2);
      assertEquals(expResult, result, 0.0);
}
```

8.3.3 测试套件 TestSuite 的应用

测试套件主要用于同时运行多个测试用例，在 JUnit 4.x 中的编写方法如下：

(1) 创建一个空类作为测试套件的入口，在 NetBeans 中的创建方法为：右击对应包，在快捷菜单选择“新建”→“其他”→“JUnit”→“测试套件”，其默认命名为 NewTestSuite.java。

(2) 使用标识@RunWith 和@SuiteClasses 修饰这个空类。

(3) 将 org.junit.runners.Suite.class 作为标识@RunWith 的参数，以提示 JUnit 为此类使用套件运行器(Runner)执行。

(4) 将需要放入此测试套件的测试类组成数组作为标识@SuiteClasses 的参数。

(5) 保证这个空类使用 public 修饰，而且存在公开的不带有任何参数的构造函数。

JUnit 4.x 中创建的 TestSuite 框架如下：

```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;
@RunWith(Suite.class)
@SuiteClasses({calculator.MyMathTest.class, calculator.MyMath2Test.class})
public class NewTestSuite {
}
```

要运行测试套件，在 NetBeans 中右击创建的测试套件类，在快捷菜单中选择“测试文件”命令，即可将对应的测试包中的所有测试用例同时进行测试。

8.3.4 参数化测试

在单元测试中，可能需要不同区域或不同类型的测试数据以验证程序的功能，这时可以采用两种方法：一是创建若干个测试函数，分别传入参数，例如上面的计算器中的 testDiv()

函数，可以创建 testDiv1()，testDiv2()，……；二是可以利用参数化测试来进行集中测试。相比较而言，第一种方法不如第二种方法简单、高效。下面来介绍参数化测试，其测试步骤如下：

（1）为准备使用参数化测试的测试类指定特殊的运行器 org. junit. runners. Parameterized。

（2）为测试类声明几个变量，分别用于存放期望值和测试所用数据。

（3）为测试类声明一个使用注解 org. junit. runners. Parameterized. Parameters 修饰的，返回值为 java. util. Collection 的公共静态方法，并在此方法中初始化所有需要测试的参数对。

（4）为测试类声明一个带有参数的公共构造函数，并在其中为步骤(2)中声明的几个变量赋值。

（5）编写测试方法，使用定义的变量作为参数进行测试。

针对以上的 MyMath. java 类中的 add()方法，可以测试不同类型的数据在程序中的效果，可编写以下参数化类：

```
package calculator;

import java.util.Arrays;
import java.util.Collection;
import org.junit.Assert;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import org.junit.runners.Parameterized.Parameters;
@RunWith(Parameterized.class)      //为下面的参数化类 ParameterTest 指定特殊运行器
public class ParameterTest {
    private float param1;          //参数 1
    private float param2;          //参数 2
    private float target;          //期望值
    @Parameters                   //参数准备函数,以@Parameters 标识,返回值为 Collection 类型
    public static Collection prepareParameters(){
        return Arrays.asList(new Object[][]{ {1,1,2},{2.1,3.2,5.3},{4,3,8}});
        //在该静态方法中声明所有需要测试的参数对,以数组形式存放
}
//公共构造函数,在该函数中为前面定义的几个变量赋值
public ParameterTest(float param1,float param2,float target){
        this.param1=param1;
        this.param2=param2;
        this.target=target;
    }
    @Test                          //测试方法,使用前面定义的变量作为参数进行测试
    public void testParametersMethod(){
        Assert.assertEquals(target, MyMath.add(param1, param2),0);
}
```

运行测试文件,发现有一个测试失败,原因是第三组测试数据中期望值设置为8,实际测试结果为7。

JUnit主要是对模块进行反复测试,尤其是在模块被修改之后。目前,JUnit衍生出了很多xUnit,例如针对Web的HtmlUnit、针对.NET的NUnit、针对C++语言的CppUnit和针对Delphi的DUnit等,从而极大地方便了测试人员的测试工作。

8.4 HtmlUnit测试

HtmlUnit是JUnit的扩展测试框架之一,该框架模拟浏览器的行为,开发者可以使用其提供的API对页面的元素进行操作。HtmlUnit"是Java程序的浏览器"。项目可以模拟浏览器运行,被誉为Java浏览器的开源实现。这个没有界面的浏览器运行速度非常迅速。HtmlUnit支持HTTP、HTTPS和COOKIE,也支持表单的POST和GET方法,能够对HTML文档进行包装,页面的各种元素都可以被当作对象进行调用,另外对JavaScript的支持也比较好。

HtmlUnit依然要采用JUnit的测试框架,但是首先需要将HtmlUnit测试jar包添加到项目中,可从http://sourceforge.net/projects/htmlunit/files/网站下载HtmlUnit测试jar包,如htmlunit-2.11。下面以Eclipse为例,介绍HtmlUnit的应用。

8.4.1 添加jar包到项目中

第一步,将下载的HtmlUnit压缩包进行解压。

第二步,在项目上右击,在弹出的快捷菜单中依次选择Build Path→Configure Build Path,打开如图8.5所示的项目属性窗口,并选中Libraries选项卡。

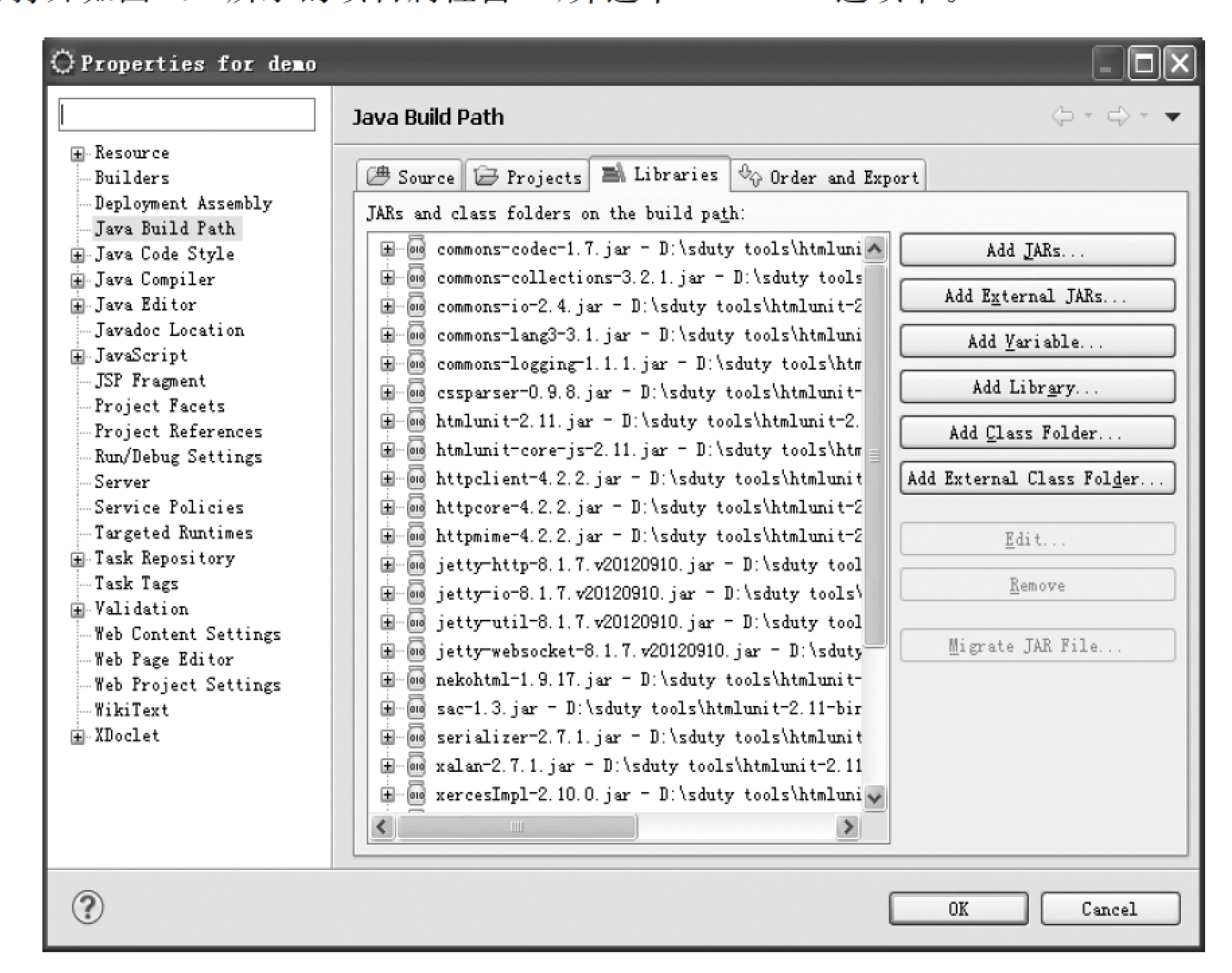


图8.5 项目属性窗口

第三步，单击窗口右侧的 Add External JARs 按钮，选择 htmlunit-2.11 中 lib 文件夹下要添加的 jar 包，并单击 OK 按钮，可将 HtmlUnit 测试所依赖的相关 jar 包添加到项目中，如图 8.6 所示。

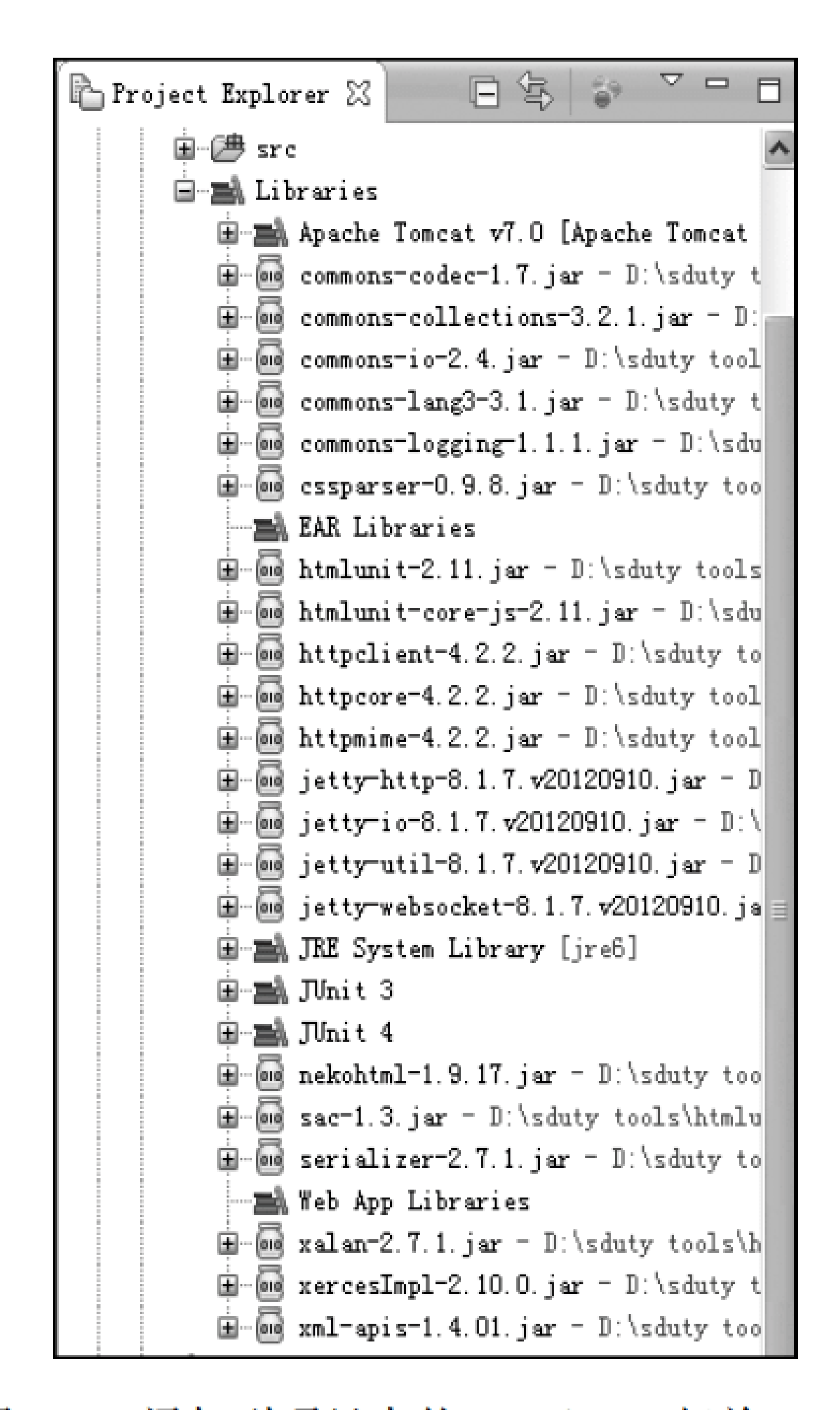


图 8.6　添加到项目中的 HtmlUnit 相关 jar 包

8.4.2　HtmlUnit 的应用

1. 载入页面

HtmlUnit 已经封装好了 HTTP 请求的方法，对于请求一个页面，举例如下：

```
import com.gargoylesoftware.htmlunit.FailingHttpStatusCodeException;
import com.gargoylesoftware.htmlunit.WebClient;
import com.gargoylesoftware.htmlunit.html.HtmlPage;
  public static void getHomeTitle() throws FailingHttpStatusCodeException,
  MalformedURLException, IOException{
  final WebClient webClient =new WebClient();
  final HtmlPage htmlPage =webClient.getPage("http://htmlunit.sourceforge.net ");
  System.out.println(htmlPage.getTitleText());
  System.out.println(htmlPage.getTextContent());
}
```

这就是一类比较常用的形式，主要用于获取 HtmlUnit 网站的 title。WebClient 是一个浏览器对象，含有多种浏览器上可进行的操作方法，如 getPage 方法。该函数就是通过 URL 取得要访问的页面。getPage 返回的文档被转化为 HtmlPage 对象，也就是被包装为 HTML 格式的对象，该对象可以输出页面的内容、标题或者一个表格等。如下面两段代码可通过 WebClient 获得一个特定浏览器版本和特定 ID 的 DIV。

示例一：获取特定浏览器版本。

```
//获取一个特定的浏览器版本
  public void homePage_Firefox() throws Exception {
  final WebClient webClient = new WebClient(BrowserVersion.FIREFOX_2);
  final HtmlPage page = webClient.getPage("http://htmlunit.sourceforge.net");
  assertEquals("HtmlUnit - Welcome to HtmlUnit", page.getTitleText());
}
```

示例二：获取特定 ID 的 DIV。

```
//获取特定 ID 的 DIV
public void getElements() throws Exception {
  final WebClient webClient = new WebClient();
  final HtmlPage page = webClient.getPage("http://some_url");
  final HtmlDivision div = page.getHtmlElementById("some_div_id");
  final HtmlAnchor anchor = page.getAnchorByName("anchor_name");
}
```

2. 模拟用户登录

HtmlUnit 对 JavaScript 的支持不是很完善，有时载入 JavaScript 就会报错，在不影响被测功能的前提下，可以通过 client. setJavaScriptEnabled(false) 的方法设置当前 JavaScript 为无效。为了方便模拟用户登录程序的执行，验证码最好设置为一个固定值，接着执行一个 POST 请求即可，代码片段如下：

```
URL url = new URL("http://www.baidu.com");
WebRequestSettings reqSet = new WebRequestSettings(url,SubmitMethod.POST);
List reqParam = new ArrayList();
reqParam.add(new NameValuePair("entered_login", username));
reqParam.add(new NameValuePair("entered_password", password));
reqParam.add(new NameValuePair("entered_imagecode", verifycode));
reqSet.setRequestParameters(reqParam);
HtmlPage mypage = (HtmlPage)client.getPage(reqSet);
```

通过上述代码可以看出对 HTTP 请求的实现方式，通过 SubmitMethod 来指定 POST 或者 GET 方法类型，将请求的参数赋给 WebRequestSettings 对象，最后以 getPage 方法将请求发送给服务器。

3. 对 Ajax 函数的测试

在介绍具体的细节问题之前，先说明整个测试框架的结构。测试工具一方面读取 case 文件，通过自己编写的 SQL 语句从数据源取得期望结果；另一方面将 case 文件中的测试数据组装为请求参数，通过 HtmlUnit 将这些 HTTP 请求发送给服务器；对 Ajax 函数的调用，没有通过模拟页面的 click 事件来进行，而是自定义一些 JavaScript 函数，在 JavaScript 函数中调用被测 Ajax 函数。

通过编程的方法，并且避过页面的操作来对 Ajax 进行测试时需要保证 Ajax 函数改为同步方法，在 DWR 这种 Ajax 框架下，DWREngine. setAsync(false)表示该函数是同步

的，只有得到服务器回应才执行其后的其他函数。这么做只是为了方便地得到测试执行结果。

自定义的JavaScript函数统计写在一个JSP或者HTML页面中，并将这个页面放在被测程序的服务器目录下，JavaScript函数举例如下：

```
function GetAccocunts() {
    DWREngine.setAsync(false);
    mtlDWR.GetAccocunts(GetAccountsCallback);
}
function GetAccocuntsCallback(data) {
    document.getElementById("resultText").value =data.toString();
}
```

有了上述一段JavaScript代码，就可以利用HtmlUnit来进行JavaScript函数的执行并从页面元素中得到实际测试结果。代码举例如下：

```
private String getActResult(HtmlPage page,String jsFunctionStr,String
reqParamValue) throws Exception{
    page.getElementById("param").setAttribute("value", reqParamValue);
    page.executeJavaScript(jsFunctionStr);
    String resultStr =page.getElementById("resultText").
    getAttribute("value").toString();          //得到回调函数的返回值
    return resultStr;
}
```

这段代码中，HtmlPage首先将写满了自定义JavaScript函数的页面载入，然后通过getElementById()方法获取页面元素，使用这种页面元素作为传递参数的暂存地，并且HtmlPage还可以用executeJavaScript()直接运行页面上指定函数名的JavaScript函数，运行完毕之后，从自定义的页面上取出实际测试结果。

4. 对HTML元素的操作

对于非Ajax的功能点，会有一些请求的返回结果写在页面的HTML标签内，此时就可以利用HtmlUnit对页面元素的操作来做一些工作。例如一个表单提交之后，返回的结果写在页面的html标签中，取出其中的结果的代码如下所示：

```
//从html标签中的list=取出相应字段
String actResultStr = page. getElementsByTagName ( " body"). item (0). getAttributes ( ).
getNamedItem("list").getNodeValue();
```

page是HtmlPage类的对象实例，HtmlUnit提供很多不同的方法使得开发人员能够获得页面的不同内容，比如常见的得到一个表单、一个锚点或一个元素。如下所示，这个类也是从DomNode类继承而来的，也包含了对XPath的操作。

```
java.lang.Object
com.gargoylesoftware.htmlunit.html.DomNode
com.gargoylesoftware.htmlunit.SgmlPage
com.gargoylesoftware.htmlunit.html.HtmlPage
```

在某些系统中，有的页面(如编辑页面)载入的数据都是放在一个隐藏的表单之中的，并且 UE(User Experience，用户体验)介入系统开发之后，HTML 元素的命名也比较规范，标签的 ID 或者 name 一般都会存在，所以根据这些可以轻松得到页面内容，如下所示：

```
resultForm = (HtmlForm)page.getElementById("ReportSaveForm");
map.put("templateid", resultForm.getInputByName("templateid").getValueAttribute());
map.put("reqsource", resultForm.getInputByName("reqsource").getValueAttribute());
map.put("reporttype", resultForm.getInputByName("reporttype").getValueAttribute());
……
```

8.4.3 使用 HtmlUnit 过程中的一些问题

在使用 HtmlUnit 时应注意以下几个问题。

(1) 在 DWR Ajax 框架下进行 HtmlUnit 的应用时，JSP 页面会引用 DWR 映射衍生的一些 JavaScript 文件，这些外部 JavaScript 文件可能会加载失败。此时可以将 DWR 动态映射产生的 JavaScript 文件保存为静态 JavaScript 文件，在 JSP 页面内直接引用就能避免这种情况发生。

(2) 如果要测试多用户登录系统并进行操作的功能，一种方法是当一个 userid 的 case 执行完毕之后，执行一个登出的 POST 请求，然后重新模拟用户登录；另外一种方法就是重新实例化一个 WebClient 对象，类似于重新打开一个浏览器程序，使得该 userid 在新的 WebClient 对象上进行操作。

(3) 因为 HtmlUnit 是基于 HTTP 请求的，所以开发时需要明确 POST 和 GET 请求的 URL，比如提交表单的“. do”，以及 Ajax 的函数名，并且还需要明确参数。

8.5 案例分析——利用 JUnit 进行 NextDate 单元测试

下面以 NextDate 的实现为例，介绍 JUnit 单元测试。

8.5.1 问题描述及主要函数实现

NextDate 是一个实现输入 3 个参数：年(y)、月(m)、日(d)，返回输入日期后一天的那个日期(nextday，nextmonth，nextyear)的面向对象程序。通过分析发现，该程序中主要根据不同的月份进行不同的处理，如可分成 31 天大月份{1，3，5，7，8，10}、30 天小月份{4，6，9，11}、特殊月份{12}和{2}4 种情况。各种情况下又要考虑天数，如大月份 31 天，小月份 30 天，平年 28 天和闰年 29 天的不同情况。因此，该程序的逻辑结构比较复杂，程序源码如下：

```
public class ValidDate {
    public static boolean isLeap(int year){                    //判断是否闰年
        if((year % 4 ==0 && year % 100 !=0) || year % 400 ==0)
            return true ;
        else
            return false;
```

```
}
public static boolean validDayRange(int day){              //判断 day 的有效性
    if((day>=1)&&(day<=31)){
        System.out.println("The day is valid day:"+"Day="+day);
        return true;
    }else{
        if((day<1)|(day>31)){
            System.out.println("The Day is invalid day:"+"Day="+day);
            return false;
        }
    }
    return false;
}
public static boolean validMonthRange(int month){          //判断 month 是否有效
if((month>=1)&&(month<=12)){
    System.out.println("Month="+month);
    return true;
}
else
    if((month<1)|(month>12)){
        System.out.println("The Month is invalid month:"+"Month="+month);
        return false;
    }
    return false;

}
public static boolean validYearRange(int year){            //判断 year 是否有效
if((year>=1500)&&(year<=2050)){
    System.out.println("Year="+year);
    return true;
}
else
    if((year<1500)|(year>2050)){
        System.out.println("The Year is invalid year:"+"Year="+year);
        return false;
    }
return false;
}
public static boolean validCombine(int day,int month,int year){
                                                    //判断 day,month,year 的组合是否有效
    if((day==31)&&((month==2)||(month==4)||(month==6)||(month==9)||
    (month==11))){
        System.out.println("Day="+day+"can not be happened in"+month);
        return false;
    }
```

```
        if((day==30) && (month==2)){
            System.out.println("Day="+day+"can not be happened in Febrary");
            return false;
        }
        if((day==29)&&(month==2)&&!(isLeap(year))){
            System.out.println("Day="+day+"can not be happened in Febrary");
            return false;
        }
        return true;

    }
    public static boolean validate(int day,int month,int year){    //综合判断各函数的有效性
        if(!validDayRange(day))
            return false;
        if(!validMonthRange(month))
            return false;
        if(!validYearRange(year))
            return false;
        if(!validCombine(day,month,year))
            return false;
        return true;
    }

}
```

8.5.2 NextDate 问题的 JUnit 测试

要做 JUnit 测试，首先需要编写测试程序。采用之前介绍的工具及对应的方法，可自动生成如下完整的测试框架：

```
package nextdate;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author limei
 */
public class ValidDateTest {

    public ValidDateTest() {
```

```
    }

    @BeforeClass
    public static void setUpClass() throws Exception {
    }

    @AfterClass
    public static void tearDownClass() throws Exception {
    }

    @Before
    public void setUp() {
}

    @After
    public void tearDown() {
    }

    /**
     * Test of isLeap method, of class ValidDate.
     */
    @Test
    public void testIsLeap() {                          //测试 IsLeap 函数
        System.out.println("isLeap");
        int year = 0;
        boolean expResult = false;
        boolean result = ValidDate.isLeap(year);
        assertEquals(expResult, result);
        //TODO review the generated test code and remove the default call to fail.
        //fail("The test case is a prototype.");
    }

    /**
     * Test of validDayRange method, of class ValidDate.
     */
    @Test
    public void testValidDayRange() {                   //测试 ValidDayRange 函数
        System.out.println("validDayRange");
        int day = 0;
        boolean expResult = false;
        boolean result = ValidDate.validDayRange(day);
        assertEquals(expResult, result);
        //TODO review the generated test code and remove the default call to fail.
        //fail("The test case is a prototype.");
```

```
}

/**
 * Test of validMonthRange method, of class ValidDate.
 */
@Test
public void testValidMonthRange() {                          //测试 ValidMonthRange 函数
    System.out.println("validMonthRange");
    int month = 0;
    boolean expResult = false;
    boolean result = ValidDate.validMonthRange(month);
    assertEquals(expResult, result);
    //TODO review the generated test code and remove the default call to fail.
    //fail("The test case is a prototype.");
}

/**
 * Test of validYearRange method, of class ValidDate.
 */
@Test
public void testValidYearRange() {                           //测试 ValidYearRange 函数
    System.out.println("validYearRange");
    int year = 0;
    boolean expResult = false;
    boolean result = ValidDate.validYearRange(year);
    assertEquals(expResult, result);
    //TODO review the generated test code and remove the default call to fail.
    //fail("The test case is a prototype.");
}

/**
 * Test of validCombine method, of class ValidDate.
 */
@Test
public void testValidCombine() {                             //测试 ValidCombine 函数
    System.out.println("validCombine");
    int day = 0;
    int month = 0;
    int year = 0;
    boolean expResult = false;
    boolean result = ValidDate.validCombine(day, month, year);
    assertEquals(expResult, result);
    //TODO review the generated test code and remove the default call to fail.
    //fail("The test case is a prototype.");
}

/**
```

```
     * Test of validate method, of class ValidDate.
     */
    @Test
    public void testValidate() {                           //测试 Validate 函数
        System.out.println("validate");
        int day = 0;
        int month = 0;
        int year = 0;
        boolean expResult = false;
        boolean result = ValidDate.validate(day, month, year);
        assertEquals(expResult, result);
        //TODO review the generated test code and remove the default call to fail.
        //fail("The test case is a prototype.");
    }

}
```

通过以上的测试框架，建立起被测程序和JUnit测试框架之间的联系，在此测试框架基础上，可对各个测试函数进行修改，如可将IsLeap测试函数修改为如下代码：

```
public void testIsLeap() {
    System.out.println("isLeap");
    //int year = 0;
    //boolean expResult = false;
    boolean result1 = ValidDate.isLeap(1600);
    boolean result2 = ValidDate.isLeap(2000);
    boolean result3 = ValidDate.isLeap(2014);
    boolean result4 = ValidDate.isLeap(2049);
    assertEquals(true, result1);
    assertEquals(true, result2);
    assertEquals(false, result3);
    assertEquals(false, result4);
    //TODO review the generated test code and remove the default call to fail.
    //fail("The test case is a prototype.");
}
```

在修改后的测试代码中，主要采用assertEquals方法，可同时测试多组被测数据的有效性。

8.6 本章小结

本章主要介绍了白盒测试方法。首先简单介绍了白盒测试工具的分类。其次，主要介绍了JUnit和HtmlUnit测试的方法，在JUnit测试中，介绍了基本的JUnit测试框架及测试方法；在掌握基本JUnit测试方法后，进一步介绍了JUnit的高级应用，包括限时测试、测试异常、参数化测试和测试套件的应用。最后，以一个完整的NextDate问题为例，演示了JUnit测试方法的实际应用过程。

习　题

1. JUnit 中提供了哪些断言？每种断言的作用是什么？

2. JUnit 中常用的接口和类有哪些？各个接口和类的含义是什么？

3. 采用 Java 语言实现一个简单的登录程序，并使用 JUnit 提供的断言、接口及高级应用中的相关方法完成 JUnit 测试。

4. 试比较 JUnit 3.X 和 JUnit 4.X 的异同。

第 9 章　性能测试案例分析

众所周知,对于电信计费软件而言,每月 20 日左右是市话交费的高峰期,全市几千个收费网点同时启动。收费过程一般分为两步,首先要根据用户提出的电话号码来查询出其当月产生费用,然后收取现金并将此用户修改为已交费状态。一个用户看起来简单的两个步骤,但当成百上千的终端同时执行这样的操作时,情况就大不一样了,如此众多的交易同时发生,对应用程序本身、操作系统、中心数据库服务器、中间件服务器和网络设备的承受力都是一个严峻的考验。决策者不可能在发生问题后才考虑系统的承受力,因此,预见软件的并发承受力,这是在软件测试阶段就应该解决的问题。而性能测试就是通过自动化的测试工具模拟多种正常、峰值以及异常负载条件来对系统的各项性能指标进行测试。负载测试、压力测试和容量测试都属于性能测试,三者可以结合进行。通过负载测试,确定在各种工作负载下系统的性能,目标是测试当负载逐渐增加时,系统各项性能指标的变化情况。压力测试是通过确定一个系统的瓶颈或者不能接受的性能点,来获得系统能提供的最大服务级别的测试。容量测试即在系统全部资源"满负荷"的情形下,测试系统的承受能力,目的是检查被测系统处理大量数据的能力。

总之,虽然负载测试、压力测试和容量测试的目的不同,但其手段、方法均比较相似,在测试过程中,通常会采用相同的测试工具及测试环境,且都会监控系统所占用资源的情况及相应的性能指标。本章在介绍性能测试的相关概念和工具的基础上,以 LoadRunner 为例主要介绍性能测试工具的使用。

9.1　性能测试概述

9.1.1　性能测试的目的

性能测试的目的是验证软件系统是否能够达到用户提出的性能指标,同时发现软件系统中存在的性能瓶颈,优化软件,最终起到优化软件系统的目的。

具体来说,性能测试的目的主要有以下几个方面:

(1) 评估系统的能力。在性能测试中,测试所得到的相关数据,如响应时间等可以被用于验证所计划的模型的能力,并帮助作出决策。

(2) 识别系统中的弱点。受控的负荷可以被增加到一个极端的水平,并突破它,从而修复系统的瓶颈或薄弱的地方。

(3) 系统调优。重复运行测试,验证调整系统的活动得到了预期的结果,从而改进性能。通过长时间的执行测试可导致程序发生由于内存泄露引起的失败,可揭示程序中的隐含问题。

(4) 验证系统稳定性及可靠性。在一定负荷下执行测试一定的时间,是评估系统稳定性和可靠性是否满足要求的唯一方法。

9.1.2 性能测试的准备

为了保证性能测试结果的可靠性以及性能测试的顺利进行，在进行性能测试前，应该做好充分的准备工作，具体如下。

1. 测试环境

配置测试环境是测试实施的一个重要阶段，测试环境的适合与否会严重影响测试结果的真实性和正确性。测试环境包括硬件环境和软件环境，硬件环境指测试必需的服务器、客户端、网络连接设备以及打印机/扫描仪等辅助硬件设备所构成的环境；软件环境指被测软件运行时的操作系统、数据库及其他应用软件构成的环境。

一个充分准备好的测试环境有三个优点：一个稳定、可重复的测试环境，能够保证测试结果的正确；保证达到测试执行的技术需求；保证得到正确的、可重复的以及易理解的测试结果。

2. 测试工具

并发性能测试是在客户端执行的黑盒测试，一般不采用手工方式，而是利用工具采用自动化方式进行。目前，成熟的并发性能测试工具有很多，选择的依据主要是测试需求和性价比。著名的并发性能测试工具有 QALoad、LoadRunner、Benchmark Factory 和 Webstress 等。这些测试工具都是自动化负载测试工具，通过可重复的、真实的测试，能够彻底地度量应用的可扩展性和性能，可以在整个开发生命周期、跨越多种平台、自动执行测试任务，可以模拟成百上千的用户并发执行关键业务而完成对应用程序的测试。

3. 测试数据

在初始的测试环境中需要输入一些适当的测试数据，目的是识别数据状态并且验证用于测试的测试案例，在正式的测试开始以前对测试案例进行调试，将正式测试开始时的错误降到最低。在测试进行到关键过程环节时，非常有必要进行数据状态的备份。制造初始数据意味着将合适的数据存储下来，需要的时候恢复它，初始数据提供了一个基线，用来评估测试执行的结果。

在测试正式执行时，还需要准备业务测试数据，比如测试并发用户数等，那么要求对应的数据库和表中有相当的数据量以及数据的种类应能覆盖全部业务。

4. 模拟真实环境测试

有些软件，特别是面向大众的商品化软件，在测试时常常需要考察在真实环境中的表现。如测试杀毒软件的扫描速度时，硬盘上布置的不同类型文件的比例要尽量接近真实环境，这样测试出来的数据才有实际意义。

9.2 性能测试工具及网站分类介绍

性能测试一般要借助于自动化测试工具，目前有很多性能测试的工具，一般可以简单地划分为负载压力测试工具、资源监控工具、故障定位工具和调优工具。其实在实际的性能测试工具中，一种工具往往集成了多种类型的应用。常用的性能测试工具有 QALoad、LoadRunner、WebRunner、SilkPerformer、WAS（Web Application Stress tool，免费工具）等；另外还有很多性能测试的网站，如 http://tools.pingdom.com/、http://gtmetrix.

com/、http://loadimpact.com/等。性能测试工具众多，在此不能一一讲述，下面对性能测试相关的部分工具及网站做简要介绍。

9.2.1 性能测试工具

1. QALoad

QALoad 是 Compuware 公司的客户/服务器系统、企业资源配置(ERP)和电子商务应用的自动化负载测试工具。QALoad 是 QACenter 性能版的一部分，它通过可重复的、真实的测试能够彻底地度量应用的可扩展性和性能。QACenter 汇集完整的跨企业的自动测试产品，专为提高软件质量而设计。QACenter 可以在整个开发生命周期跨越多种平台自动执行测试任务。

QALoad 的主要功能如下：

(1) 预测系统性能。当应用升级或者新应用部署时，负载测试能帮助确定系统是否能按计划处理用户负载。QALoad 并不需调用最终用户及其设备，它能够仿真数以千计的用户进行业务交易。通过 QALoad，用户可以预知业务量接近投产后真实水平时端对端的响应时间，以便满足投产后的服务水平要求。

(2) 通过重复测试寻找瓶颈问题。QALoad 录制/回放能力提供了一种可重复的方法来验证负载下的应用性能，可以很容易地模拟数千个用户，并执行和运行测试。利用 QALoad 反复测试可以充分地测试与容量相关的问题，快速确认性能瓶颈并进行优化和调整。

(3) 从控制中心管理全局负载测试。QALoad Conductor 工具为定义、管理和执行负载测试提供了一个中心控制点。Conductor 通过执行测试脚本，管理无数的虚拟用户。Conductor 可以自动识别网络中可进行负载测试的机器，并在这些机器之间自动分布工作量，以避免网络超载。从 Conductor 自动启动和配置远程用户，跨国机构可以进行全球负载测试。在测试过程中，Conductor 还可以在负载测试期间收集有关性能和时间的统计数据。

(4) 验证应用的可扩展性。出于高可扩展性的设计考虑，QALoad 包括了远程存储虚拟用户响应时间并在测试结束后或其他特定时间下载这些资料的功能。这种方法可以增加测试能力，减少进行大型负载测试时的网络资源耗费。QALoad 采用轮询法采集响应时间，在不影响测试或无需增加测试投资的条件下，就可了解测试中究竟出现了什么情况。

(5) 快速创建仿真的负载测试。准确仿真复杂业务的进行，对于预测电子商务应用软件的功能至关重要。运用 QALoad，可以迅速创建出一些实际的测试方案，而不需要手工编写脚本或有关应用中间软件的详细知识和协议。

QALoad 的体系结构如图 9.1 所示。

在图 9.1 中，中心控制器用于产生 session 文件(其中保存了 player 信息和 dll 信息)，执行之，并将 Player 产生的 timming 文件交给 Analyze 进行分析；虚拟用户生成器用于执行 dll 文件，并产生一个 timming 文件；分析器用于生成各种报表，把测试结果展现给用户。

2. LoadRunner

LoadRunner 是一种较高规模适应性的自动负载测试工具，它能预测系统行为，优化性能。LoadRunner 强调的是整个企业的系统，它通过模拟实际用户的操作行为和实行实时性能监测来帮助用户更快的确认和查找问题。此外，LoadRunner 能支持最宽泛的协议和技术，为用户的特殊环境量身定做地提供解决方案。

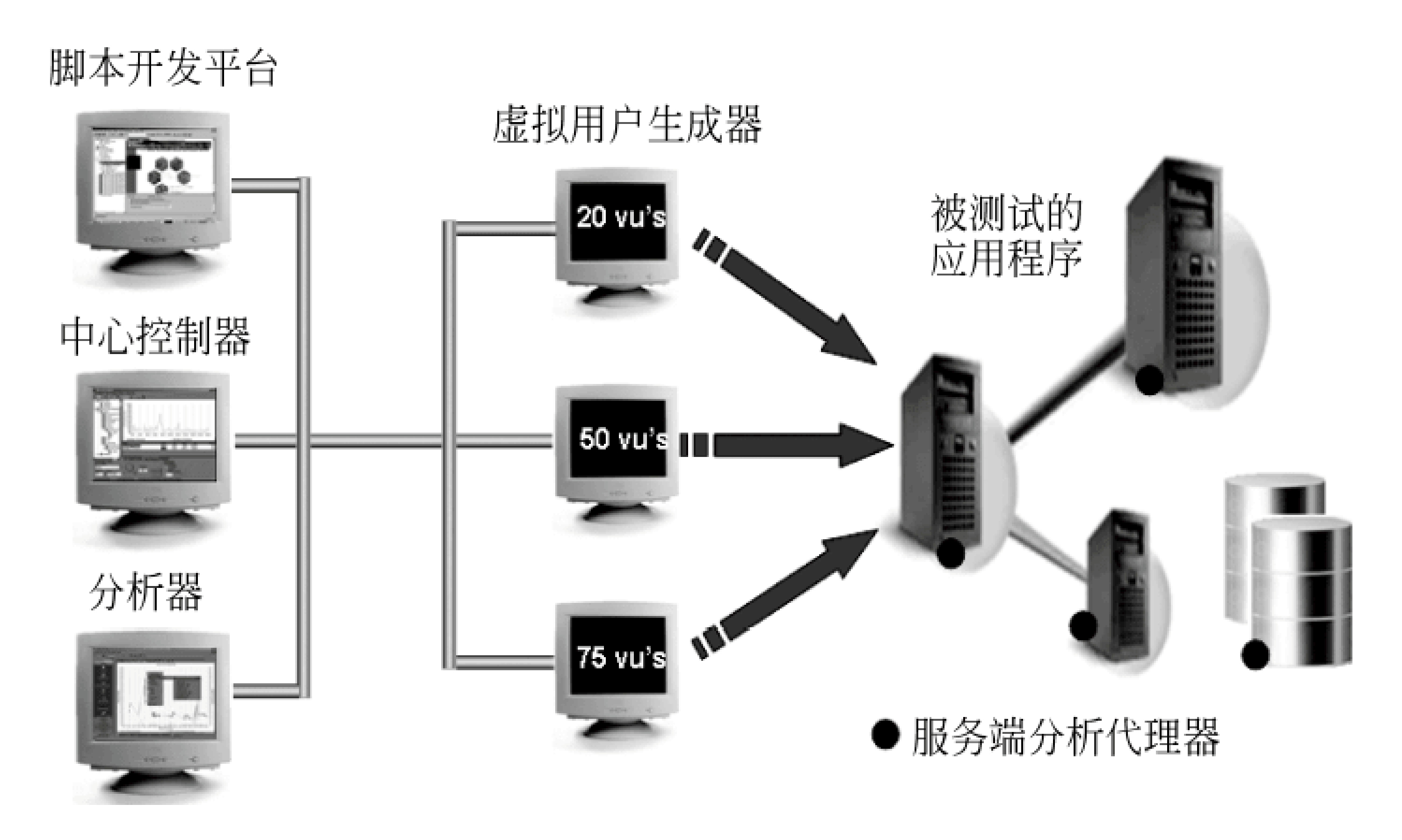


图 9.1　QALoad 的体系结构

LoadRunner 的主要功能如下：

1）轻松创建虚拟用户

使用 LoadRunner 的 Virtual User Generator，用户能很方便地创立起系统负载。该引擎能够生成虚拟用户，以虚拟用户的方式模拟真实用户的业务操作行为。它先记录下业务流程(如下订单或机票预订)，然后将其转化为测试脚本。利用虚拟用户，用户可以在 Windows、UNIX 或 Linux 计算机上同时产生成千上万个用户访问。所以 LoadRunner 能极大地减少负载测试所需的硬件和人力资源。

2）创建真实的负载

虚拟用户建立起来以后，需要设定负载方案、业务流程组合和虚拟用户数量。用 LoadRunner 的 Controller，能很快组织起多用户的测试方案。

而且，利用它的日程计划服务可以定义用户在什么时候访问系统以产生负载。这样，就能将测试过程自动化。

3）定位性能问题

LoadRunner 内含集成的实时监测器，在负载测试过程的任何时候，用户都可以观察到应用系统的运行性能。这些性能监测器为用户实时显示交易性能数据(如响应时间)和其他系统组件(包括应用服务器、Web 服务器、网络设备和数据库等)的实时性能，以帮助测试人员在测试过程中从客户和服务器两方面评估这些系统组件的运行性能，从而更快地发现问题。

利用 LoadRunner 的 ContentCheck，可以判断负载下的应用程序功能正常与否。ContentCheck 在虚拟用户运行时，检测应用程序的网络数据包内容，从中确定是否有错误内容传送出去。它的实时浏览器帮助用户从终端用户角度观察程序性能状况。

4）分析结果以精确定位问题所在

一旦测试完毕后，LoadRunner 收集汇总所有的测试数据，并提供高级的分析和报告工具，以便迅速查找到性能问题并追溯原由。使用 LoadRunner 的 Web 交易细节监测器，可以了解到将所有的图像、框架和文本下载到每一网页上所需的时间。

5）重复测试保证系统发布的高性能

负载测试是一个重复过程。每次处理完一个出错情况，用户都需要对应用程序在相同

的方案下再进行一次负载测试,以此检验所做的修正是否改善了运行性能。

LoadRunner 完全支持 EJB 的负载测试。这些基于 Java 的组件运行在应用服务器上,提供广泛的应用服务。通过测试这些组件,用户可以在应用程序开发的早期就确认并解决可能产生的问题。

3. WebRunner

WebRunner 是 RadView 公司推出的一个性能测试和分析工具,它让 Web 应用程序开发者自动执行压力测试;WebLoad 通过模拟真实用户的操作,生成压力负载来测试 Web 的性能,用户创建的是基于 JavaScript 的测试脚本,称为议程(agenda),用它来模拟客户的行为,通过执行该脚本来衡量 Web 应用程序在真实环境下的性能。

4. SilkPerformer

SilkPerformer 是 Borland 公司(原 Segue 公司)出品的企业级负载测试工具。它能够模拟成千上万的用户在多协议和多种计算环境下工作。SilkPerformer 可以让用户在使用前就能够预测企业电子商务环境的行为,不受电子商务应用规模和复杂性影响。可视的用户界面、负载条件下可视化的内容校验、实时的性能监视和强大的管理报告可以帮助用户迅速将问题隔离,通过最小化测试周期、优化性能以及确保可伸缩性,加快了投入市场的时间,并保证了系统的可靠性。

5. WAS

WAS 是 Microsoft 公司提供的免费的 Web 负载压力测试工具,应用广泛。WAS 可以通过一台或者多台客户机模拟大量用户的活动。WAS 支持身份验证、加密和 Cookies,也能够模拟各种浏览器和调制解调器速度,它的功能和性能可以与数万美元的产品媲美。

9.2.2 性能测试网站

目前,性能测试的网站也不少,其基本功能也都类似,如常用的网站有 http://tools.pingdom.com/, http://gtmetrix.com/, http://loadimpact.com/等,下面就以 http://tools.pingdom.com/为例简要介绍性能测试网站的工作过程。

http://tools.pingdom.com/网站是一个免费的性能测试网站,其目的是帮助用户测试所设计网站的速度,以采取有效措施优化网站的性能。该网站主要测试分三大块:测试页面的加载时间,测试 DNS 服务器和域名的设置,测试与服务器的网络连接情况。

该性能测试网站的主要功能如下:

(1) 测试一个网页中的所有内容。查看文件的大小、加载时间以及 Web 页面中的每个元素的详细情况(如 HTML、JavaScript、CSS 及图片等)。用户可以不同的方式排列组合这些元素以获取性能瓶颈。

(2) 获取网站的总体性能表现。通过测试结果,将大量相关性能统计数据进行自动相加,以提供给用户网站总体性能表现。

(3) 性能等级和技巧。从 Google Page Speed(类似于雅虎的 Yslow)页面查看被测网站是否符合最佳性能指标,并且可以获得提高网站速度的相关技巧。

(4) 跟踪网站性能历史。每次的测试都会被自动保存下来,在后面的测试中用户可以查看之前的每次测试,可以发现测试中的变化。

(5) 从不同的路径进行测试。如可以测试网站在欧洲、美国等的表现。

(6) 分享测试结果。该网站为用户的测试工作提供便利的同时,可以让用户将测试结果分享给朋友、同事或 Web 主机等。

性能测试网站的首页如图 9.2 所示,在输入框中输入要测试的网页的 URL,以测试网页的加载时间,分析网页的性能,并寻找瓶颈问题。



图 9.2 性能测试网站首页

9.3 利用 LoadRunner 进行负载测试

使用 Mercury LoadRunner 工具,可以创建场景,并在其中定义性能测试会话期间发生的事件。在场景中,LoadRunner 工具会在物理计算机上用虚拟用户(即 Vuser)代替真实用户。这些 Vuser 通过以可重复、可预测的方式模拟典型用户的操作,在系统上创建负载。LoadRunner 的报告和图表可以提供评估应用程序性能所需的信息。假设正在测试一个基于 Web 的旅行代理应用程序(用户可以通过它在线预订航班),并要确定多个用户同时执行相同的事务时,该应用程序将如何处理。使用 LoadRunner 工具,可以创建具有 1000 个 Vuser 的场景,并且这些 Vuser 可以同时尝试在应用程序中预订航班。

LoadRunner 的虚拟用户模拟测试如图 9.3 所示。

利用 LoadRunner 进行负载测试,一般需要如图 9.4 所示的 6 个步骤:测试计划、创建脚本、设计场景、运行场景、监视场景和分析结果。每个步骤均由一个 Mercury LoadRunner 工具组件执行,其中包括 Mercury 虚拟用户生成器(VuGen),Mercury LoadRunner Controller 和 Mercury Analysis,下面就以 LoadRunner 8.0 为例,并且该工具已经进行正确的安装,详细介绍负载测试的这几个步骤。

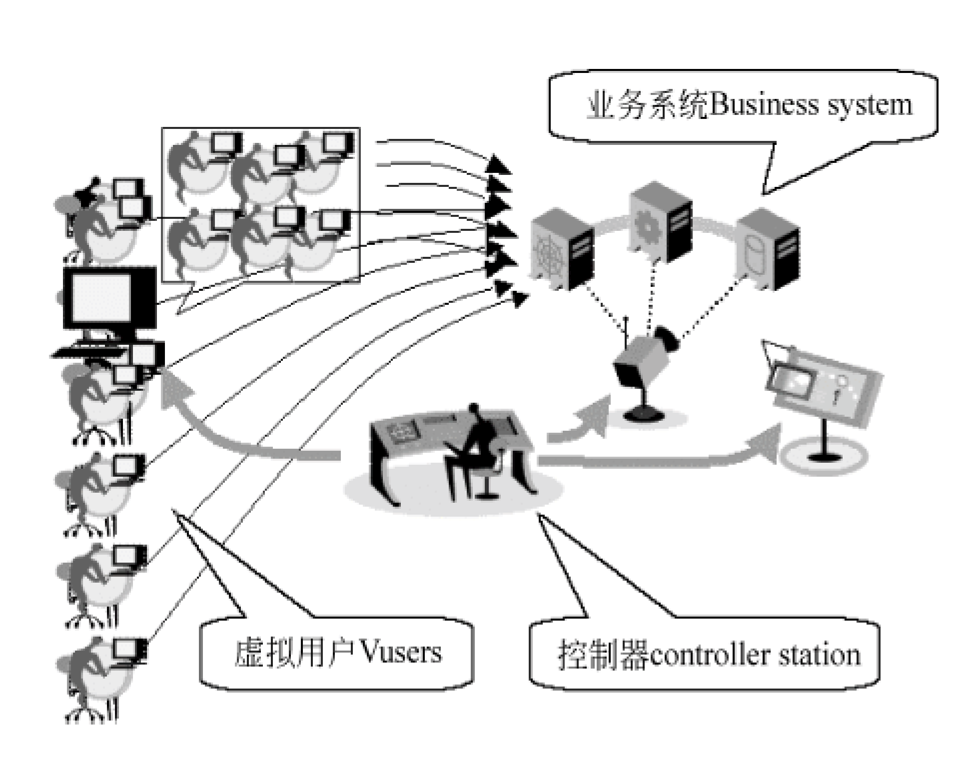


图 9.3　LoadRunner 虚拟用户模拟测试图

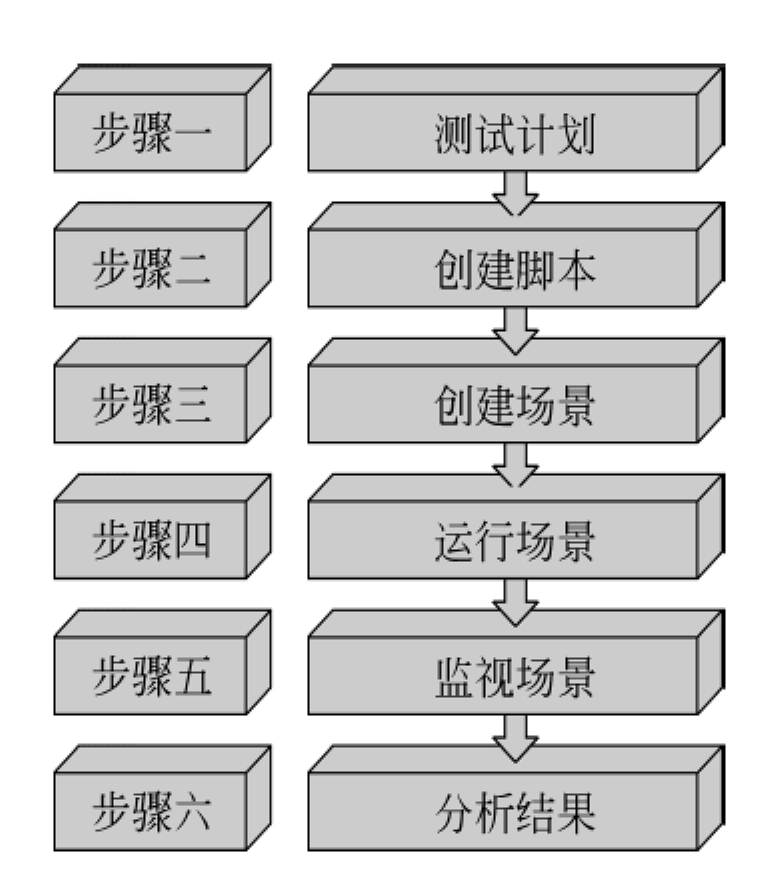


图 9.4　LoadRunner 负载测试的步骤

9.3.1　测试计划

成功的负载测试和一个好的测试计划是分不开的，清晰的测试计划可以保证按照用户设计的 LoadRunner 场景如期完成测试目标。在测试计划中，需要设计当前的虚拟用户数、典型的业务流程及用户所期望的响应时间等。一般的测试计划包括分析应用程序、定义测试目标和设计执行过程。

在分析应用程序中需要确定系统的组成部分，描述系统的配置（如连接到系统的用户数、客户机的配置、服务器数据库类型及配置、处理并发的用户数等），描述系统的功能（如划分系统功能模块、系统的用户角色、确定模块测试的优先级、系统负载的最大值等）。

在定义测试目标中需要确定操作的响应时间和系统最优的配置，检查系统的可靠性，确定软硬件的不同对系统的影响，确定系统容量及系统的性能瓶颈等系统测试目标。

设计执行过程中需要确定执行测试的过程，如软件的安装与配置，定义关键业务流程，定义任务分配，测试数据的准备，测试工具的配置，脚本的录制和调试，场景方案的设计，测试的执行，结果的收集，总结测试等。

9.3.2　脚本的录制与开发

在 LoadRunner 测试环境中，用虚拟用户代替了实际用户，虚拟用户通过重复性的以及可预期的方式模仿实际用户的行为使用系统，对系统产生负载。

LoadRunner 的 VuGen 以录制-回放的方式进行工作，当用户使用应用程序进行业务处理的过程中，VuGen 会自动地记录用户的所有操作，录制到虚拟用户脚本中。在 LoadRunner 中，脚本是负载测试的基础。

1. 启动 LoadRunner

选择“开始”→“程序”→Mercury LoadRunner→LoadRunner，即可打开 LoadRunner 主窗口，如图 9.5 所示。

2. 新建测试脚本，选择通信协议

单击主窗口中的 Create/Edit Scripts，即可打开 VuGen 欢迎窗口，如图 9.6 所示，需要选择系统通信协议。在窗口左侧的面板中可以选择 New Single Protocol Script（新建单协议脚本），New Multiple Protocol Script（新建多协议脚本），New Script Recent Protocols（新

建最近使用的协议脚本),Open Script(打开脚本)等,而且在不同的应用类型中,可选择不同的协议进行测试,如表9.1所示。

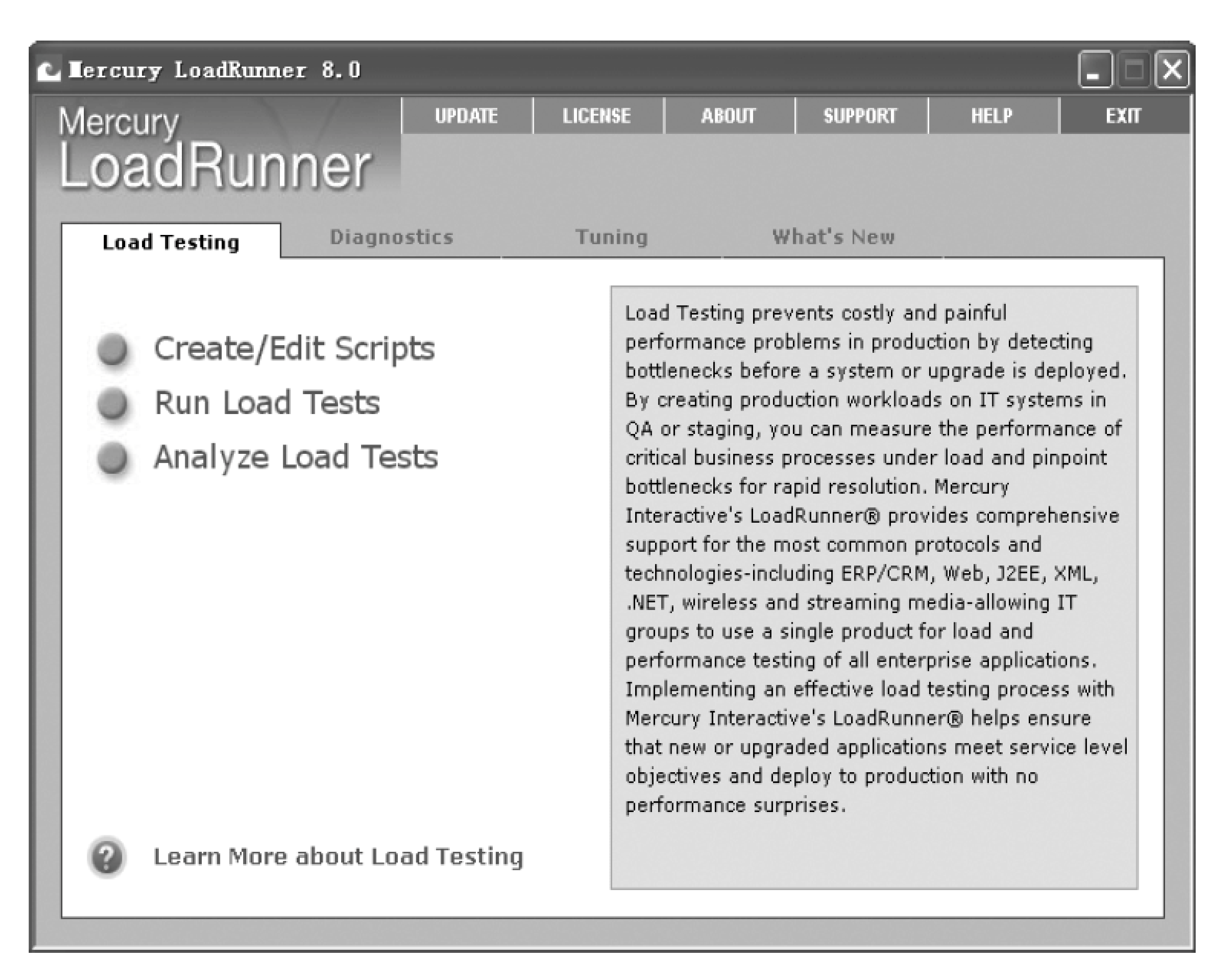


图9.5 LoadRunner主窗口

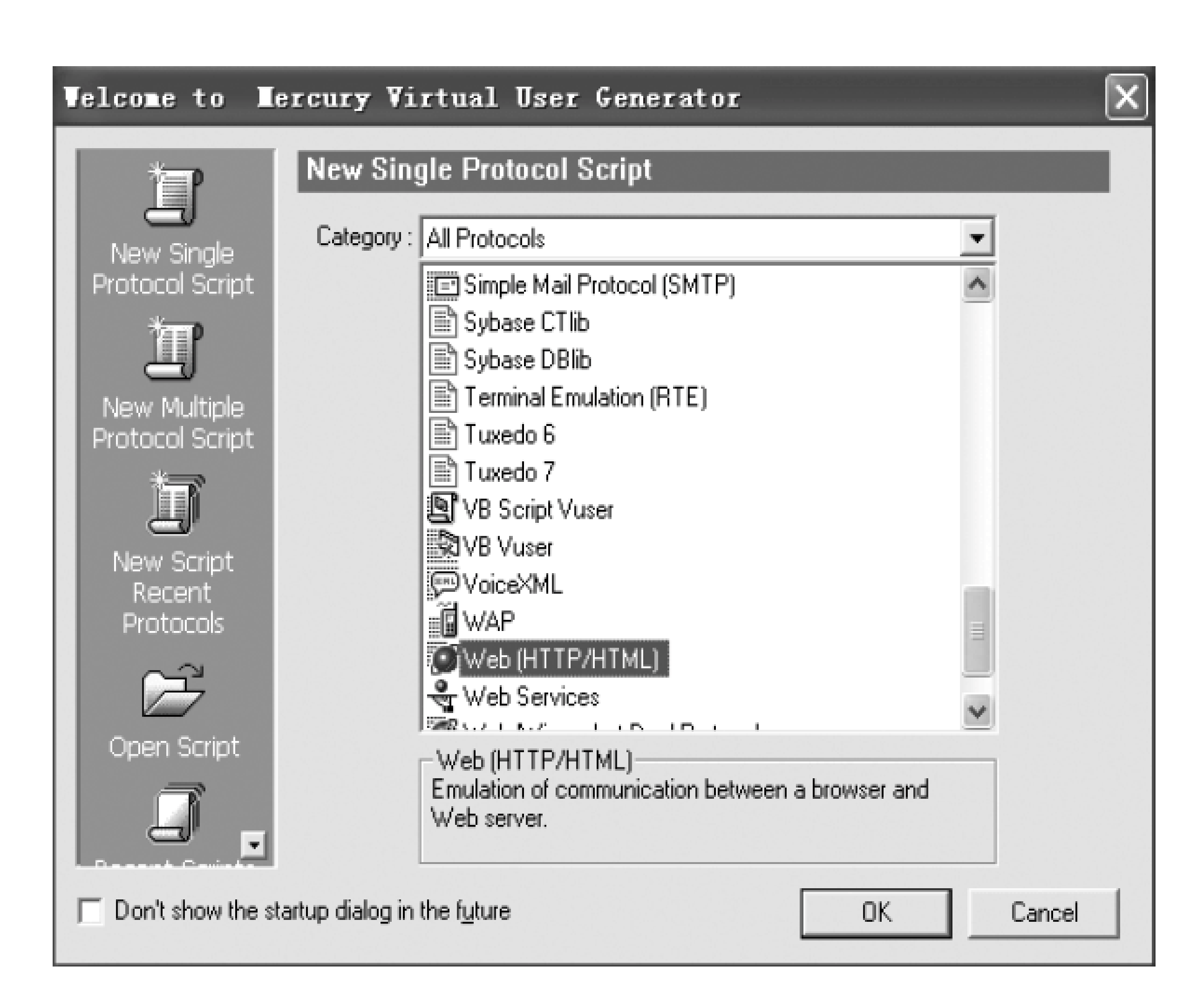


图9.6 VuGen欢迎窗口

表9.1 系统通信协议选择列表

应用类型	建议选择协议
Web网站	Web(HTTP/HTML)
FTP服务器	File Transfer Protocol(FTP)
邮件服务器	Internet Messaging Application Protocol(IMAP) Post Office Protocol(POP3) Simple Mail Transport Protocol(SMTP)

续表

应 用 类 型		建议选择协议
C/S	客户端以 ADO,OLEDB 方式连接后台数据库	Microsoft SQL Server,Oracle,Sybase,DB2,Informix
	以 ODBC 方式连接后台数据库	ODBC
	没有后台数据库	Socket
分布式组件		COM/DCOM,EJB
无线应用		WAP,Palm

在本次测试中要测试 LoadRunner 自带的预订机票的 Web 程序,因此在面板中选择 New Single Protocol Script,单击选择 Web(HTTP/HTML)协议,则会打开 VuGen 窗口,如图 9.7 所示。该窗口主要分成 3 部分:工具栏(Toolbars)、脚本步骤区(Scripts Steps)和快照区(Snapshots)。其中,左侧面板的脚本步骤区主要以树形列表显示虚拟用户的行为,在录制脚本期间,用户执行的每一个操作都会生成一个步骤(step);右边面板的快照区会显示当前所选择的步骤的快照,即在录制脚本时浏览器屏幕的一幅截图。对任何一个脚本而言,都包括 3 部分:vuser_init、Action 和 vuser_end。通过左边的小窗口可以选择快照区中显示的内容,默认显示的为 Action。通常可将登录部分放在 vuser_init 中,将登录后的操作放在 Action 中,而将关闭、注销及退出放在 vuser_end 中。

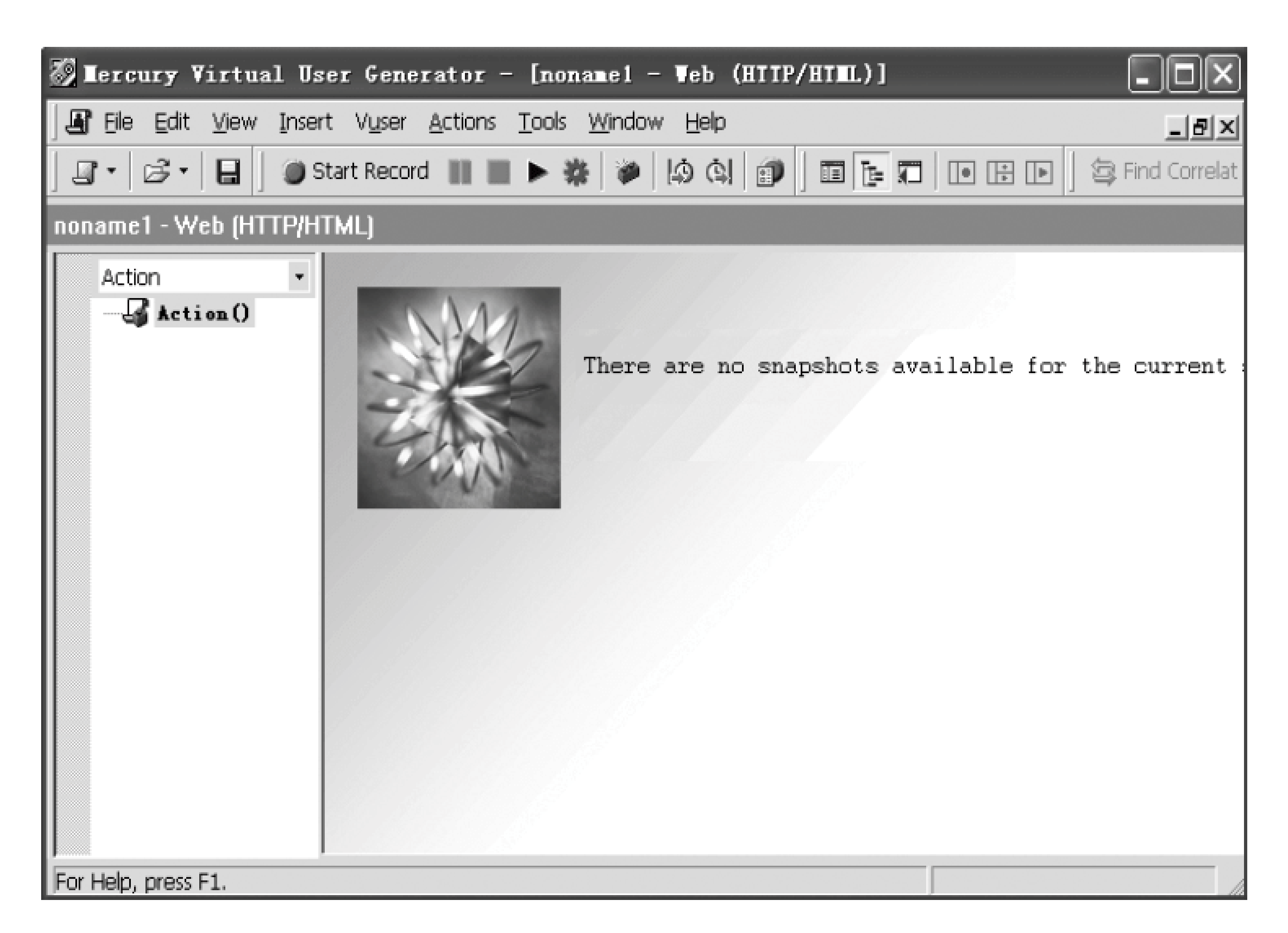


图 9.7 VuGen 窗口

3. 录制脚本(包括录制工具条介绍)

在之前的操作中,创建了一个空的 Web 脚本,下面的操作会将事件直接录制进这个空的 Web 脚本中。

步骤一,单击工具栏上的 Start Record 按钮,或者选择菜单 Vuser→Start Recording…,会打开 Start Recording 对话框,如图 9.8 所示。在该对话框的 URL 后输入 Web 页面地址,如 http://localhost:1080/mercuryWebTours,

图 9.8 Start Recording 对话框

在 Record into Action 后面选择 Action。对于 LoadRunner 而言,要测试 Web 程序,最好利用 IE 浏览器打开程序,否则可能出现不支持其他浏览器,从而导致不能录制脚本的问题。要更改浏览器,单击 Options…按钮,可打开 Recording Options 对话框,如图 9.9 所示,单击窗口左侧的 Browser 选项,选择窗口右侧的第 3 个单选按钮 Specify path to application,单击 Browse...按钮,更改浏览器。进行完必要的录制脚本设置后,单击 OK 按钮,会打开 Mercury Tours 网站。

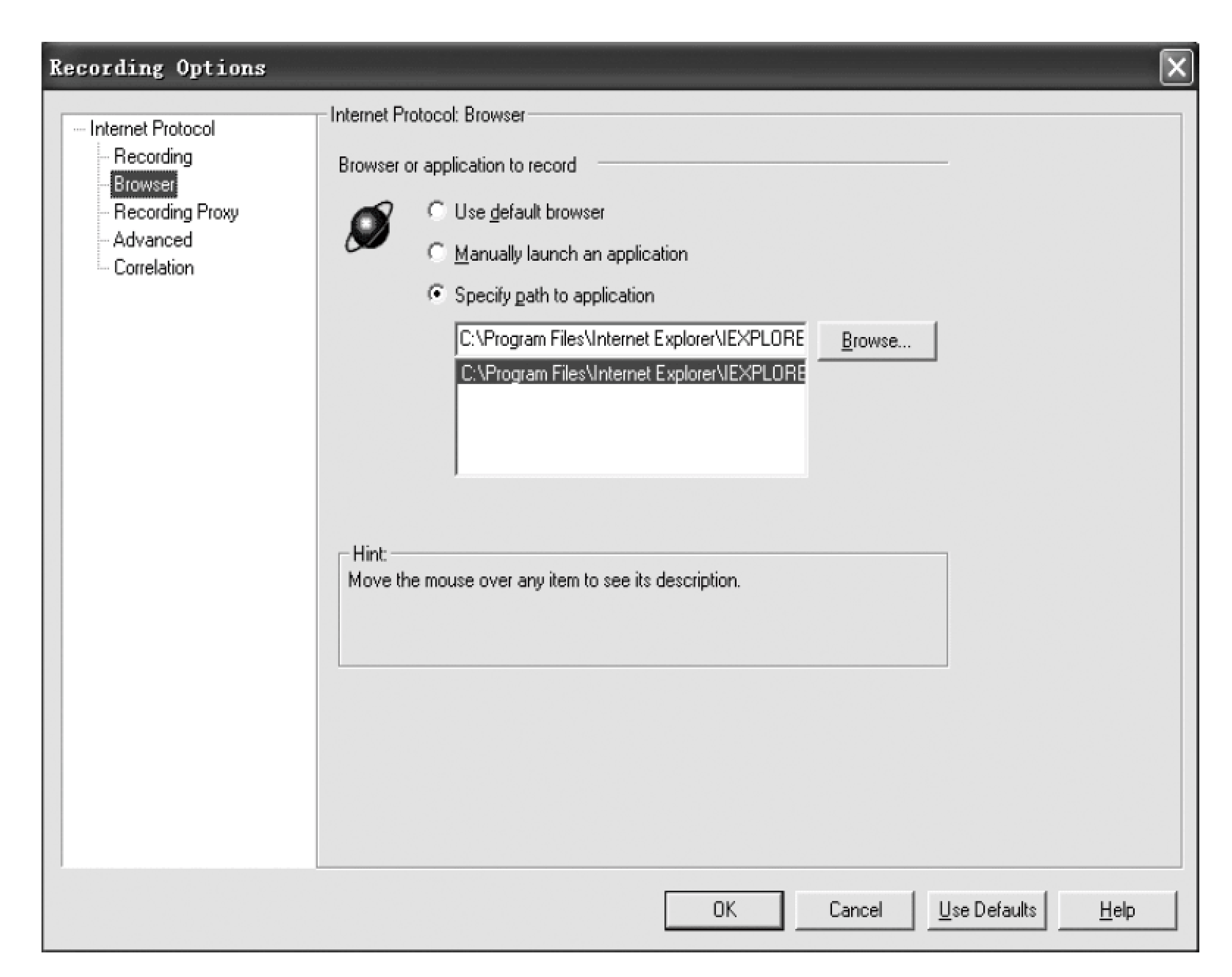


图 9.9 Recording Options 对话框

步骤二,登录到 Mercury Tours 网站并进行预订机票的操作。在用户登录 name 处输入用户名 jojo,在 password 处输入 bean。单击 login 按钮即可登录。登录之后在窗口左侧单击 flights 按钮,在窗口右侧的 Arrival City 后选择 Los Angeles 或其他目的地,其余选项保持默认或选择其他,单击 continue...按钮。进入航班搜索列表页面,不改变默认选项,单击 continue...按钮,进入支付页面,在 Credit Card 后面的编辑框输入 12345678,单击 purchase flight 按钮,进入确认预订信息页面。单击窗口左侧的 itinerary 按钮,显示所有预订信息。单击窗口左侧的 sign off 按钮,单击浮动工具条上的停止按钮,停止录制脚本。选择工具栏上的保存工具按钮,在文件名输入框后输入 basic_tutorial,单击保存按钮,即可保存该虚拟用户脚本,该脚本默认保存在 LoadRunner 下的 scripts 文件夹中,并在 VuGen 窗口的标题栏显示脚本名称。

4. 查看脚本

录制并保存脚本之后,就可以以两种方式查看脚本,即树视图和脚本视图两种方式。在录制脚本的过程中,VuGen 会为用户的每一个操作建立一个图标和标题,默认情况下,脚本会以树视图的形式显示,如图 9.10 所示。当单击树视图中的图标和标题时,根据用户需要可在 VuGen 窗口右侧的快照区域显示当时操作的页面、服务器响应信息和客户端请求信息。

如果想以脚本视图的方式查看脚本，单击工具栏上的 View Script 按钮，即可看到 vuser_int、Action 和 vuser_end 的脚本显示，在该视图中，VuGen 将虚拟用户的每个操作以函数的形式表示出来，如图 9.11 所示。

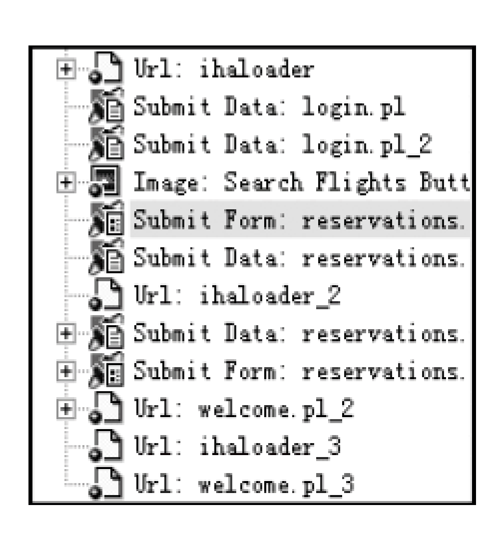


图 9.10 树视图

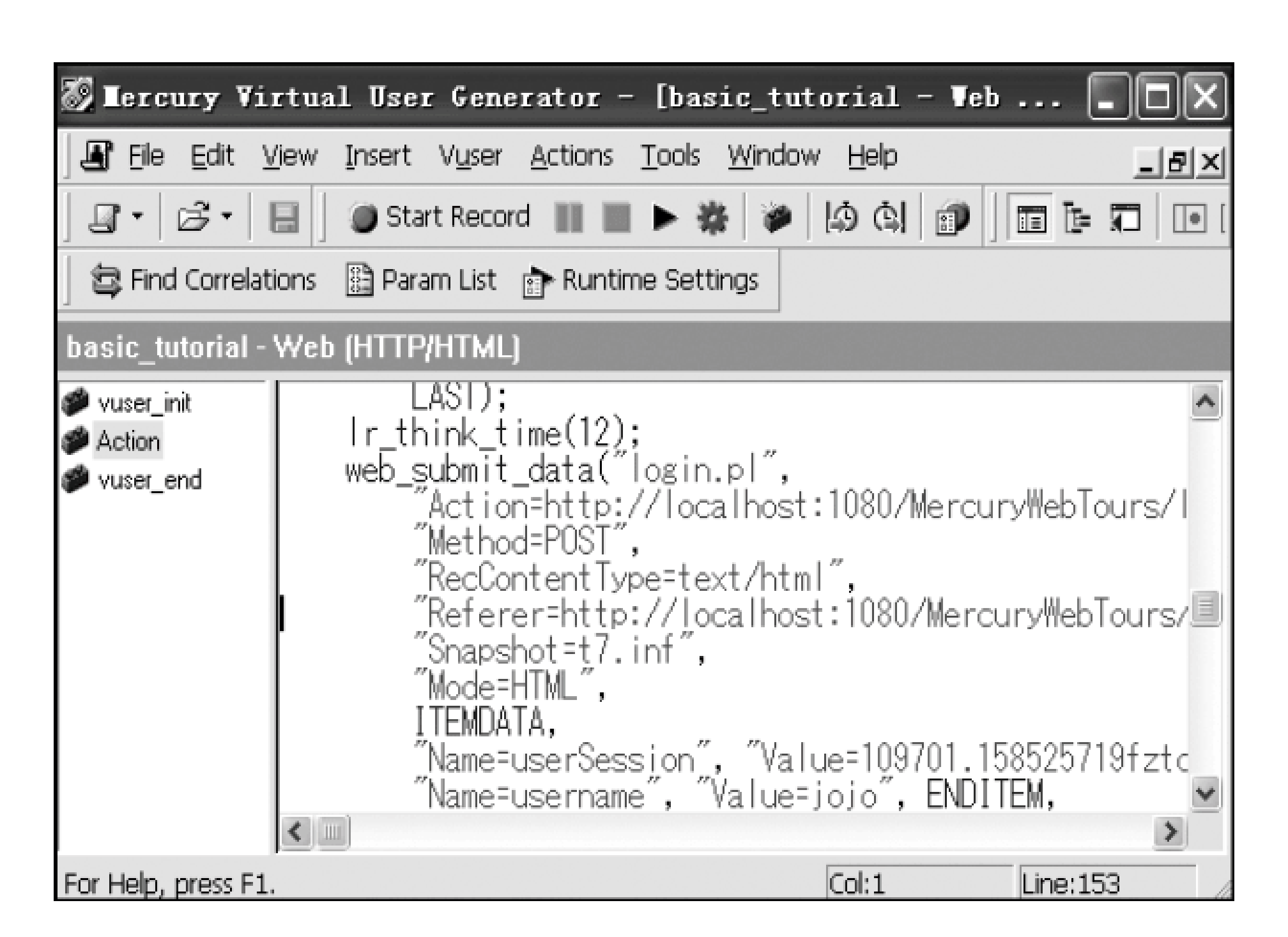


图 9.11 脚本视图

9.3.3 回放脚本

回放脚本的目的是保证用户将之前的脚本应用在所设计的负载测试的场景中时，其能够正确地按照用户的期望来进行工作。回放脚本之前，需要进行 run-time 设置，该操作可以帮助用户设置虚拟用户的行为。

1. 设置 run-time 行为

Run-time Settings 的设置包括 General、Network、Browser 和 Internet Protocol 四项。每一项里面又包括多个设置项，其中常用的有 Run Logic、Pacing、Log 和 Think Time，而其他的选项可保持默认值。其中，Run Logic 用来设置 Action 部分重复运行的次数；Pacing 用来设置以什么样的方式开始下一次重复；Log 用来设置是否启用日志以及日志的表示方式；Think Time 用于设置脚本中两个 steps 之间用户的思考时间。

单击工具栏上的 Runtime Settings 按钮，即可打开如图 9.12 所示的设置对话框。本次测试中，假如 RunLogic 中的重复次数设置为 4，在 Pacing 中选择第三项：随机数，60～90 秒，Log 选项可设置为 Extended Log 并选择 Parameter Substitution，Think Time 不做任何修改，保持默认选项。单击 OK 按钮，即可完成 Run-time Setting 的设置。

2. 查看脚本在真实环境中的运行情况

一般情况下，VuGen 会在后台运行测试，用户在脚本中不会看到虚拟用户的操作。而 VuGen 的实时查看器在脚本回放时可以查看虚拟用户的操作。这个查看器并不是实际的浏览器，只是虚拟用户进行操作时的页面快照。

依次选择 VuGen 窗口中的 Tools→General Options...菜单，打开 General Options 对话框，如图 9.13 所示。选择 Display 选项面板，并选中 Show browser during replay 复选框，清除 Display report at the end of script execution 复选框，单击 OK 按钮关闭 General Options 对话框。单击 VuGen 窗口上方工具栏中的 Run 按钮或按键盘上的 F5 键，则会打开 Run-Time Viewer 对话框，等待一会，会自动运行脚本，回放之前虚拟用户的操作，如图 9.14 所示。

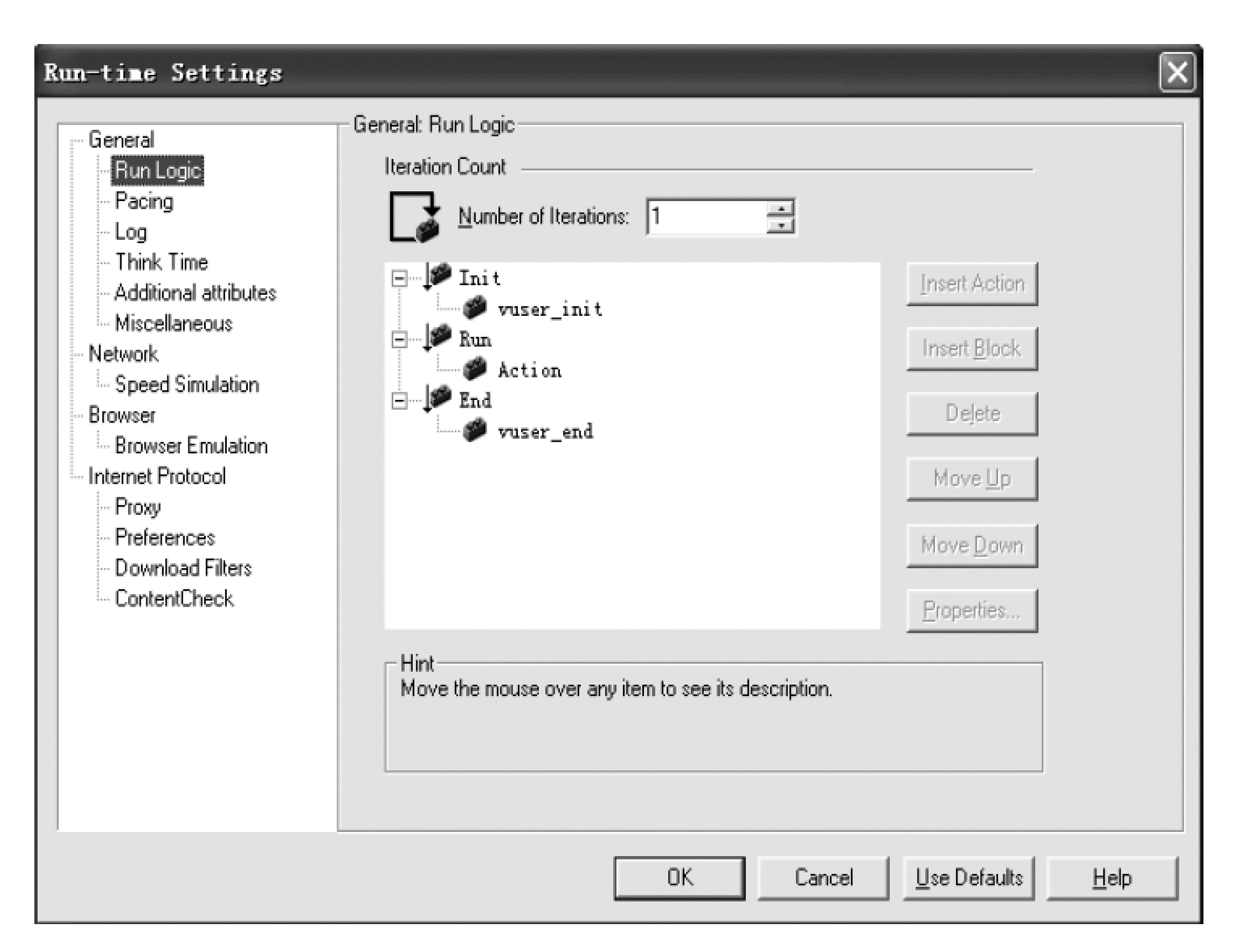


图 9.12　Run-time Settings 对话框

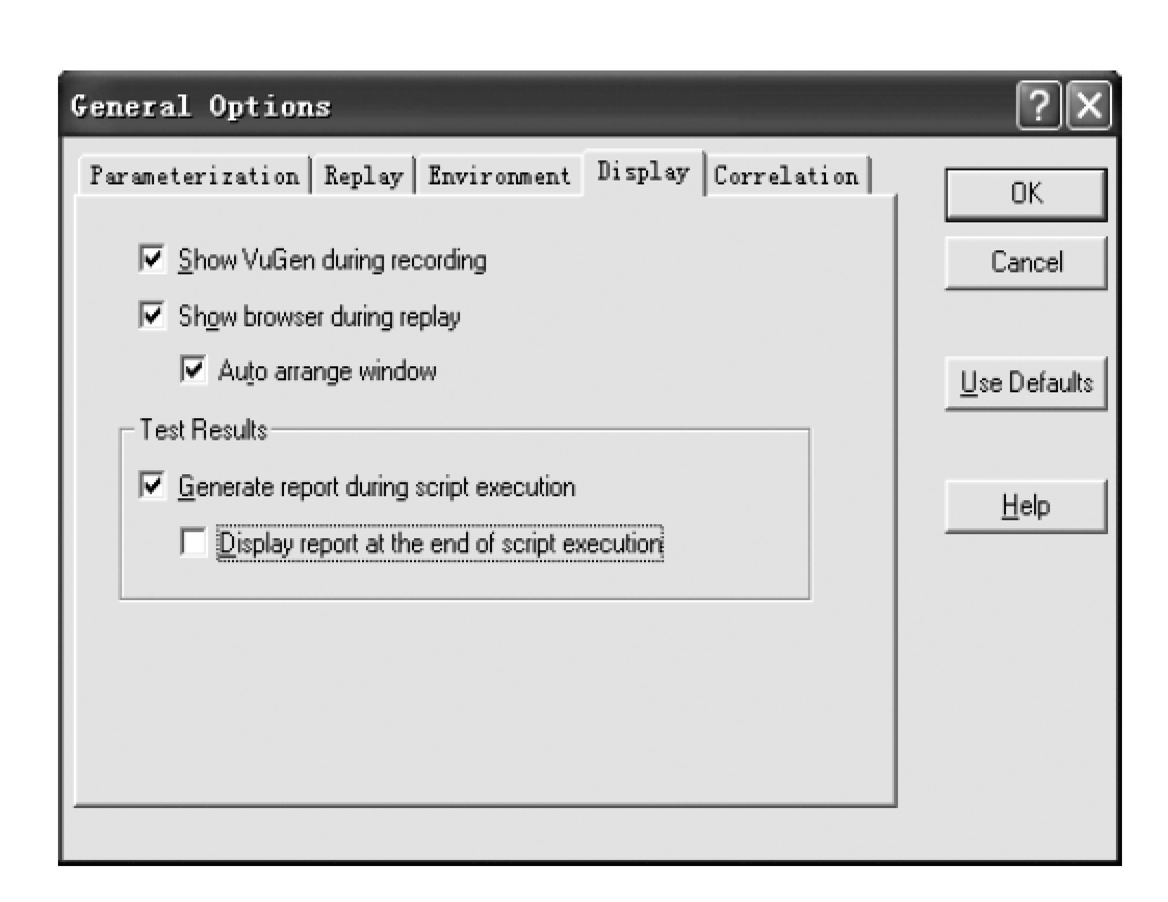


图 9.13　General Options 对话框

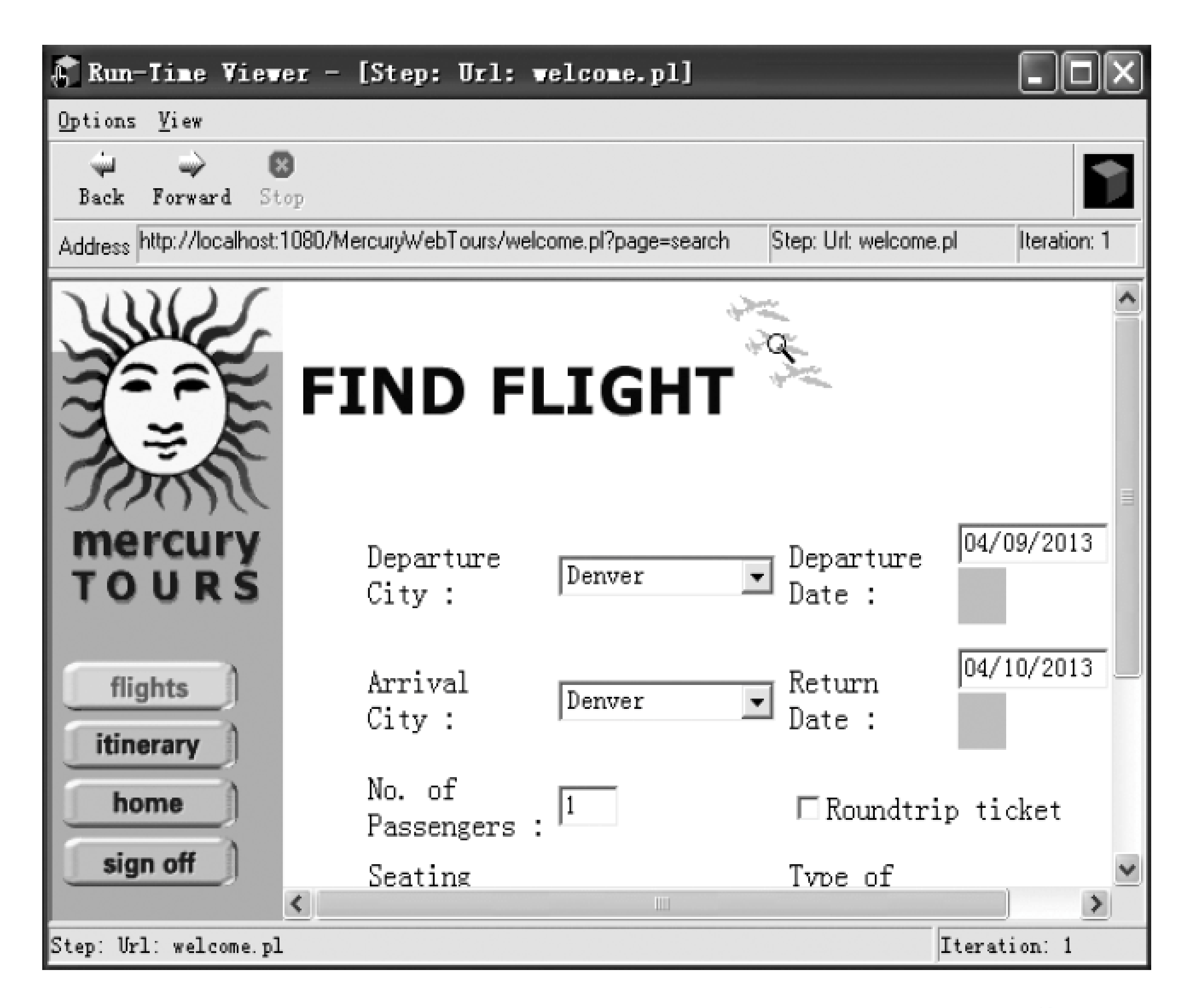


图 9.14　Run-Time Viewer 窗口中查看虚拟用户的操作

3. 查看回放事件结果

在 Run-Time Viewer 窗口中,可以看到虚拟用户的操作,如果想查看操作过程中具体的事件发生情况,可在 VuGen 窗口下方的 Execution Log 窗口中看到其文本表示形式,如图 9.15 所示。

```
Execution Log | Recording Log | Correlation Results
Action.c(158): Submitting form to "http://localhost:1080/MercuryWebTours/reservations.pl", Targ
Action.c(158): Found resource "http://localhost:1080/MercuryWebTours/images/splash_searchresult
Action.c(158): Resource "http://localhost:1080/MercuryWebTours/images/continue.gif" is in the c
Action.c(158): Found resource "http://localhost:1080/MercuryWebTours/images/startover.gif" in H
Action.c(158): web_submit_form("reservations.pl") was successful, 5821 body bytes, 567 header b
Action.c(173): Submitting form to "http://localhost:1080/MercuryWebTours/reservations.pl", Targ
Action.c(173): Found resource "http://localhost:1080/MercuryWebTours/images/splash_creditcard.g
Action.c(173): Found resource "http://localhost:1080/MercuryWebTours/images/purchaseflight.gif"
Action.c(173): Resource "http://localhost:1080/MercuryWebTours/images/startover.gif" is in the
Action.c(173): web_submit_form("reservations.pl_2") was successful, 6523 body bytes, 567 header
```

图 9.15 回放过程中的事件文本表示形式

4. 测试是否通过

在回放脚本之后,可以查看测试结果,以确定是否通过测试。可通过选择 VuGen 窗口中的菜单 View→TestResults...来打开 Test Results 窗口,如图 9.16 所示。该窗口的左侧以树形列表的形式显示测试结果,右侧以表格的形式统计测试结果。

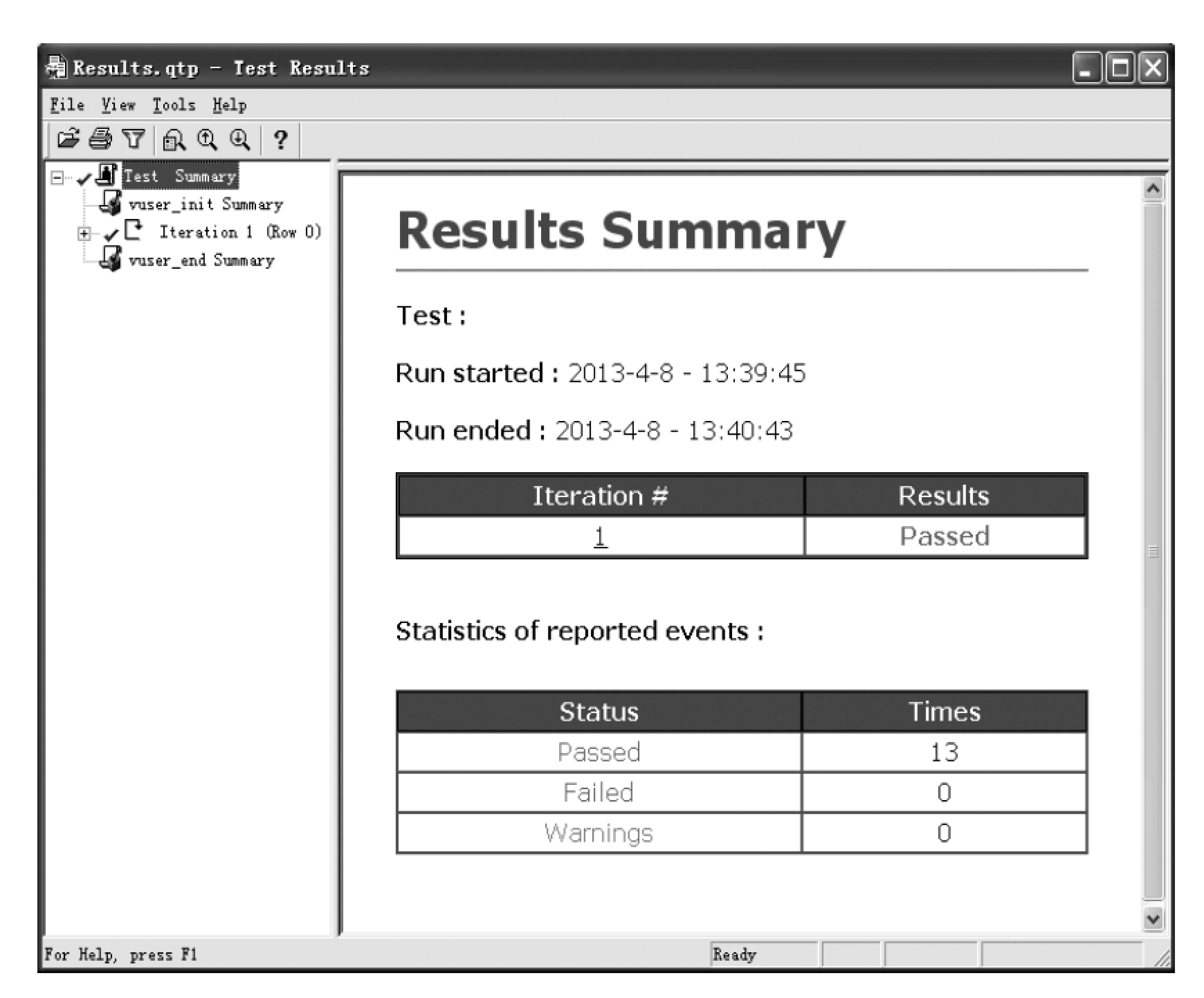


图 9.16 Test Results 窗口

5. 如何过滤结果

测试结果以统计的形式显示,查看测试结果时,如果发现有未通过(Failed)的测试,则需要查找错误的地方。最直接的操作为在 Test Results 窗口中选择 Tools→Find...菜单,则会打开 Find 对话框,如图 9.17 所示。

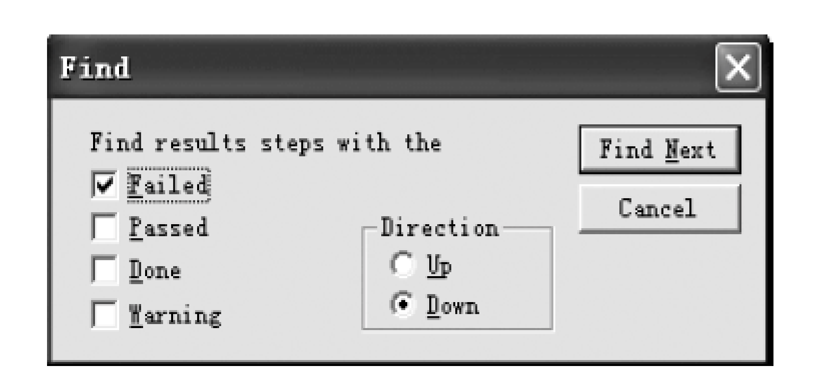


图 9.17 Find 对话框

选中 Failed 复选框,清除其他选项,单击 Find Next 按钮则会在 Test Results 窗口右侧列出测试中所有失败的测试。

9.3.4 场景设计

前面所介绍的内容主要是模拟单个用户访问系统时的业务处理流程,而 LoadRunner 的主要作用就是用于负载测试。

1. Controller 简介

负载测试主要是在特定环境下测试系统的运行情况,如同一时间多个不同用户进行同样的操作等,因此,在负载测试时需要建立强大的负载,以搭建更接近系统实际运行的环境。而 LoadRunner Controller 正好为用户提供了建立和运行负载测试的工具及环境。

要利用 Controller 进行负载测试,首先需要启动 Controller,其具体操作如下。

在图 9.5 所示的 LoadRunner 主窗口的 Load Testing 选项面板中,选择 Run Load Tests,打开 New Scenario 对话框,如图 9.18 所示。Controller 主要有两种场景类型:Manual Scenario 和 Goal-Oriented Scenario。其中,Manual Scenario 是使用手工方式建立测试场景,该场景可以设置虚拟用户的个数和虚拟用户运行的时间,还可以测试系统最大同时承受的用户的个数;Goal-Oriented Scenario 是面向目标的测试场景,该场景主要用于测试系统是否能够完成特定的目标,如系统对于特定业务的响应时间和每秒钟的业务量等,Controller 会自动地为系统建立针对目标的场景。这里选择 Manual Scenario。

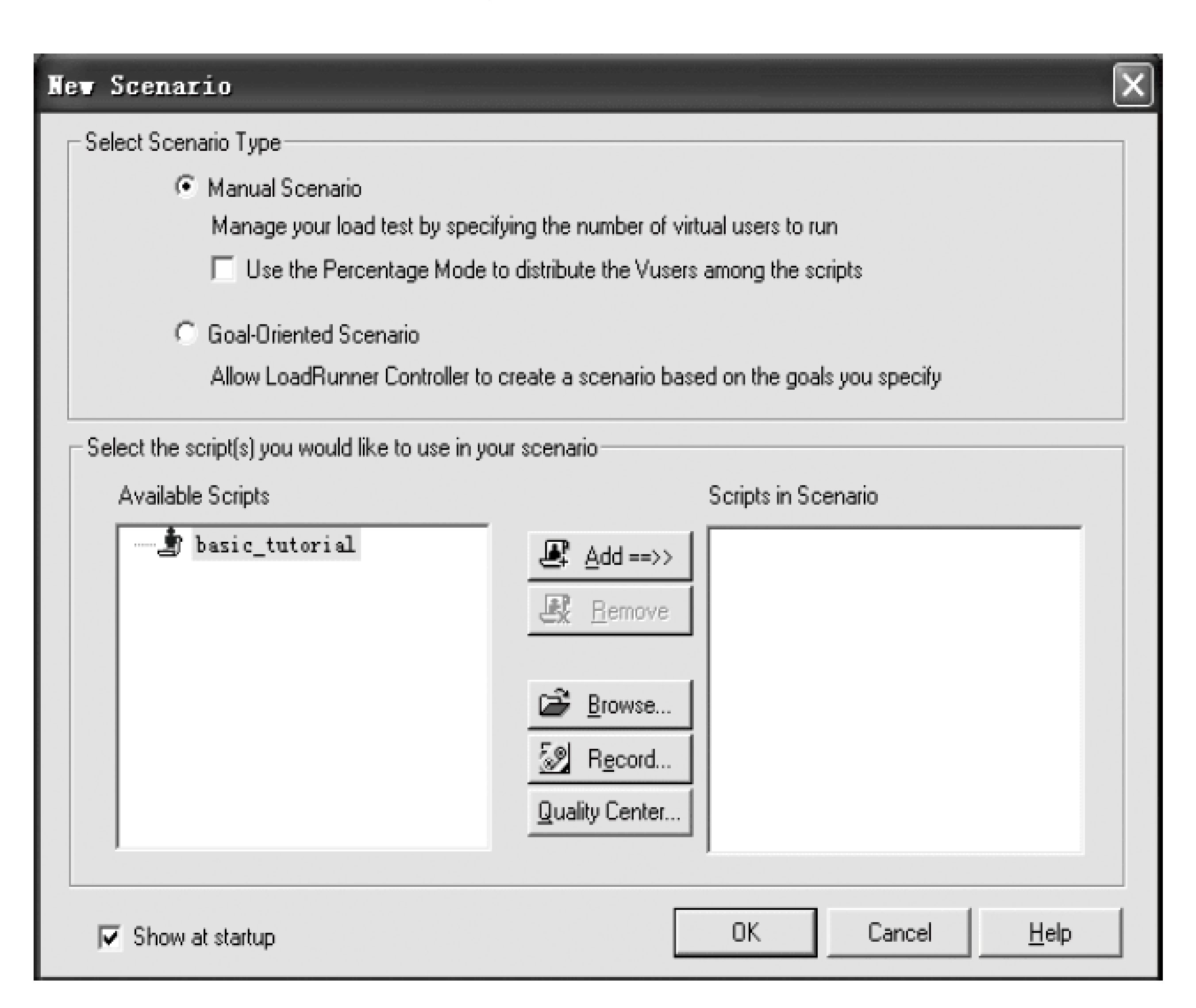


图 9.18 New Scenario 对话框

2. 添加负载测试的脚本

为了模拟不同用户使用系统的场景,需要针对不同用户的设置创建不同的场景组运行多个脚本。之前所录制的脚本中包括了用户登录、查询、买票、退出等操作。接下来,往场景中添加一个类似的脚本,并进行配置,以模仿多个用户在系统中同时进行这样的操作。这里添加 LoadRunner 自带的 basic_tutorial 脚本。单击 New Scenario 窗口中的 Browse... 按钮,从 LoadRunner 的 Scripts 文件夹中选择 basic_tutorial 脚本,单击“打开”按钮,可以看到所选择的脚本添加到了 Available Scripts 和 Scripts in Scenario 中。单击 New Scenario 窗

口中的 OK 按钮，则会打开 Scenario Design 窗口，如图 9.19 所示。该窗口由两部分组成：Scenario Schedule 和 Scenario Groups。其中，Scenario Schedule 用于设置负载的行为，如负载测试的周期、如何停止等；Scenario Groups 用于配置用户组，如添加组、定义用户的动作、虚拟用户的个数等。

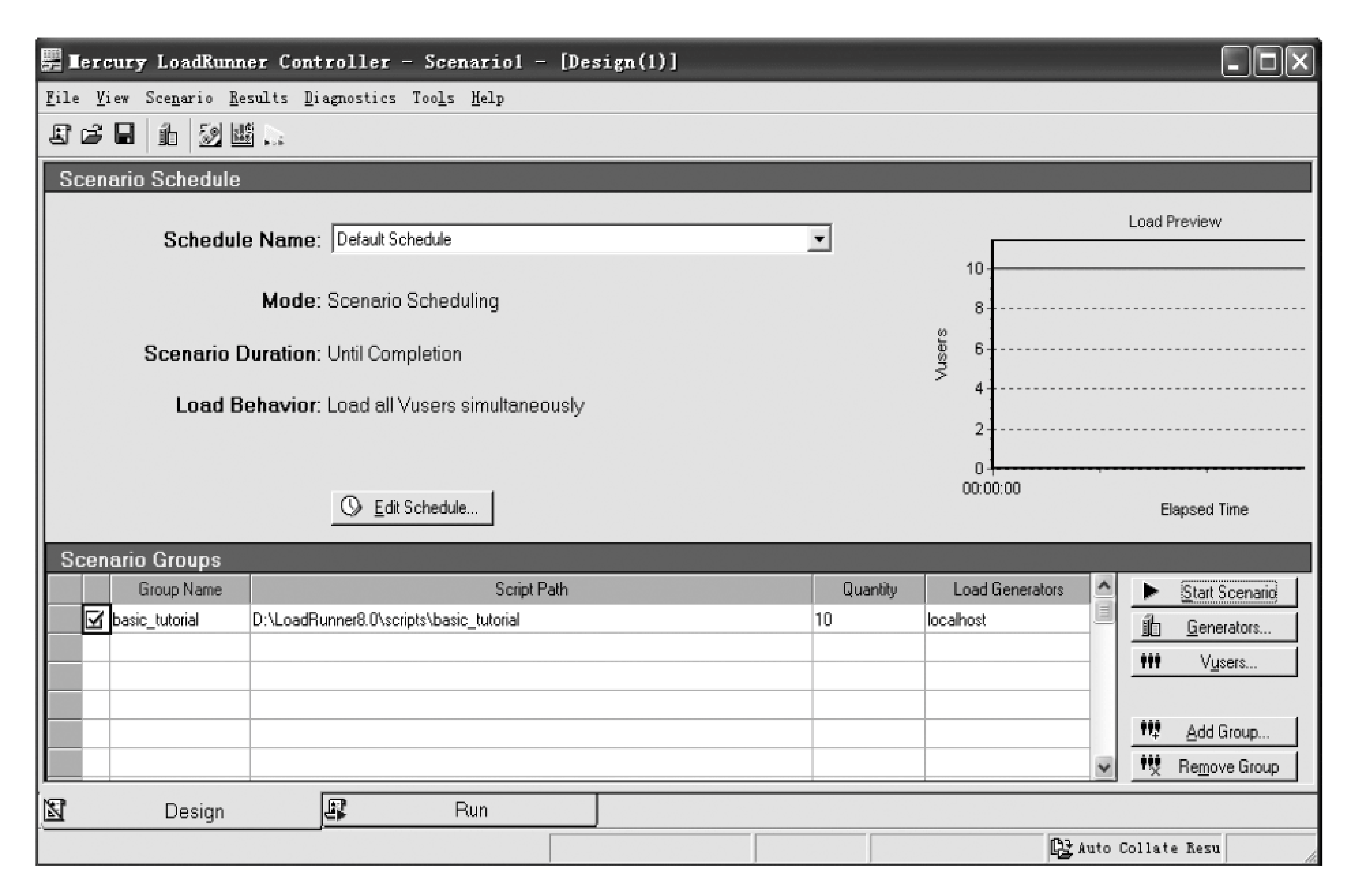


图 9.19　Scenario Design 窗口

3. 添加负载生成器

添加脚本之后，需要配置负载生成器。负载生成器通过运行虚拟用户对系统产生负载。首先需要将负载生成器添加到场景中，接着测试负载生成器的连接情况。单击图 9.19 中的 Scenario Groups 右侧的 Generators...按钮，打开 Load Generators 窗口，如图 9.20 所示。该窗口中显示本机(localhost)作为负载生成器的详细描述。其中，Status 为 Down，表示控制器和负载生成器没有连接。在该窗口中，也可以添加负载生成器。

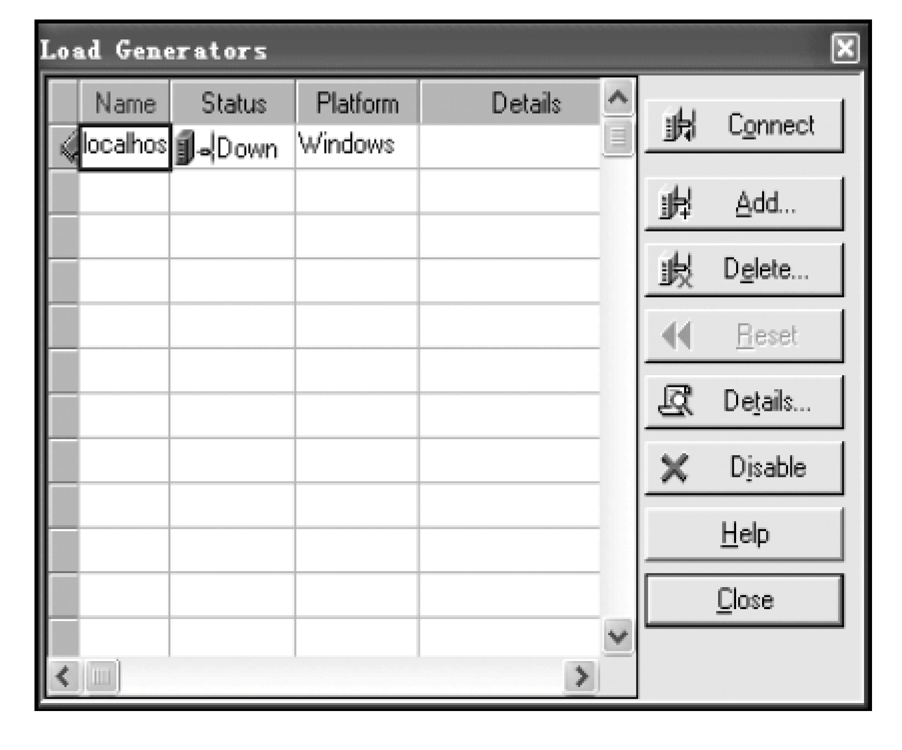


图 9.20　Load Generators 窗口

当运行场景时，控制器会自动连接到负载生成器，这里，可以先测试它们的连接情况。在 Load Generators 窗口中的生成器(如 localhost)上右击，在快捷菜单中选择 Connect，会看到 Status 为 Ready，表示连接成功。

4. 模拟实际的负载及不同类型的用户

添加负载生成器后，就可以配置负载的行为。单击 Scenario Design 窗口中 Scenario Schedule 中的 Edit Schedule...按钮，打开 Schedule Builder 对话框，如图 9.21 所示。在该对话框中可以设置 Ramp Up 中的多个用户使用系统的情况，如同时添加负载或间隔一定时间添加负载；可以设置 Duration 中虚拟用户执行业务处理的周期；还可以设置 Ramp Down 中同时停止负载或逐渐停止负载；还有其他的一些设置。这些设置可以使系统的测

试环境更接近于实际使用环境。右侧的 Load Preview 图可以预览对负载设置的结果，红色阶梯形曲线表示增加负载及取消负载的设置情况。

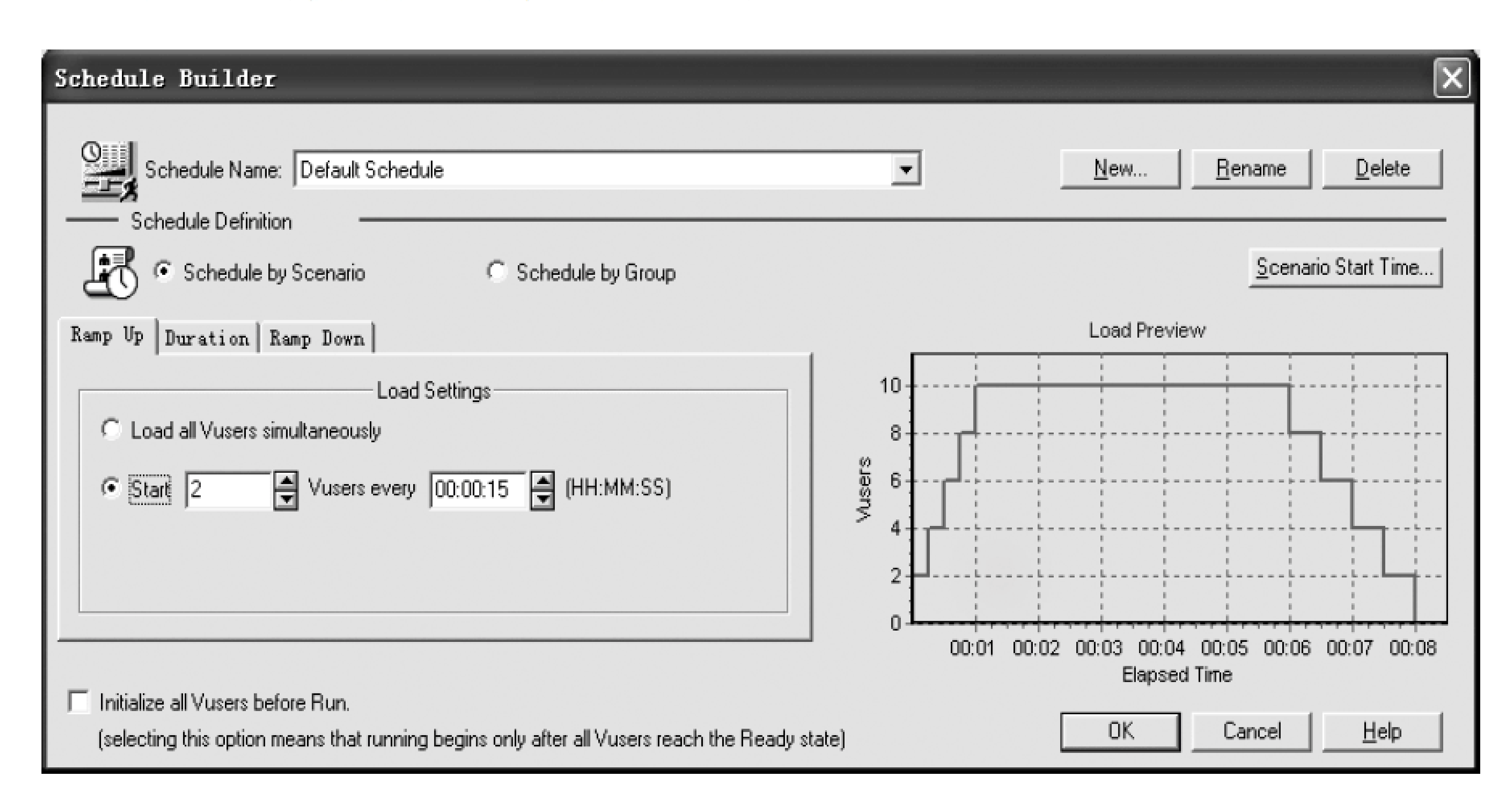


图 9.21　Schedule Builder 对话框

要模拟不同类型的用户，主要是在图 9.12 所示的 Run-time Setting 对话框中进行设置。

5. 在负载下监视系统的运行情况

设置负载行为之后，可以观察到添加负载之后的应用程序对系统的影响。这里，可以利用 LoadRunner 的监视器进行监视，例如可以看到负载对 CPU、硬盘及内存等资源的影响。单击 Controller 窗口最下方的 Run 选项面板按钮，会打开 Scenario Run 窗口，Run 视图如图 9.22 所示。如可以监视 Windows 资源、虚拟用户数、业务处理的响应时间等。还可以通过左侧的 Available Graphs 窗口打开要查看的视图。运行场景后，视图中会出现不同曲线的表示结果。

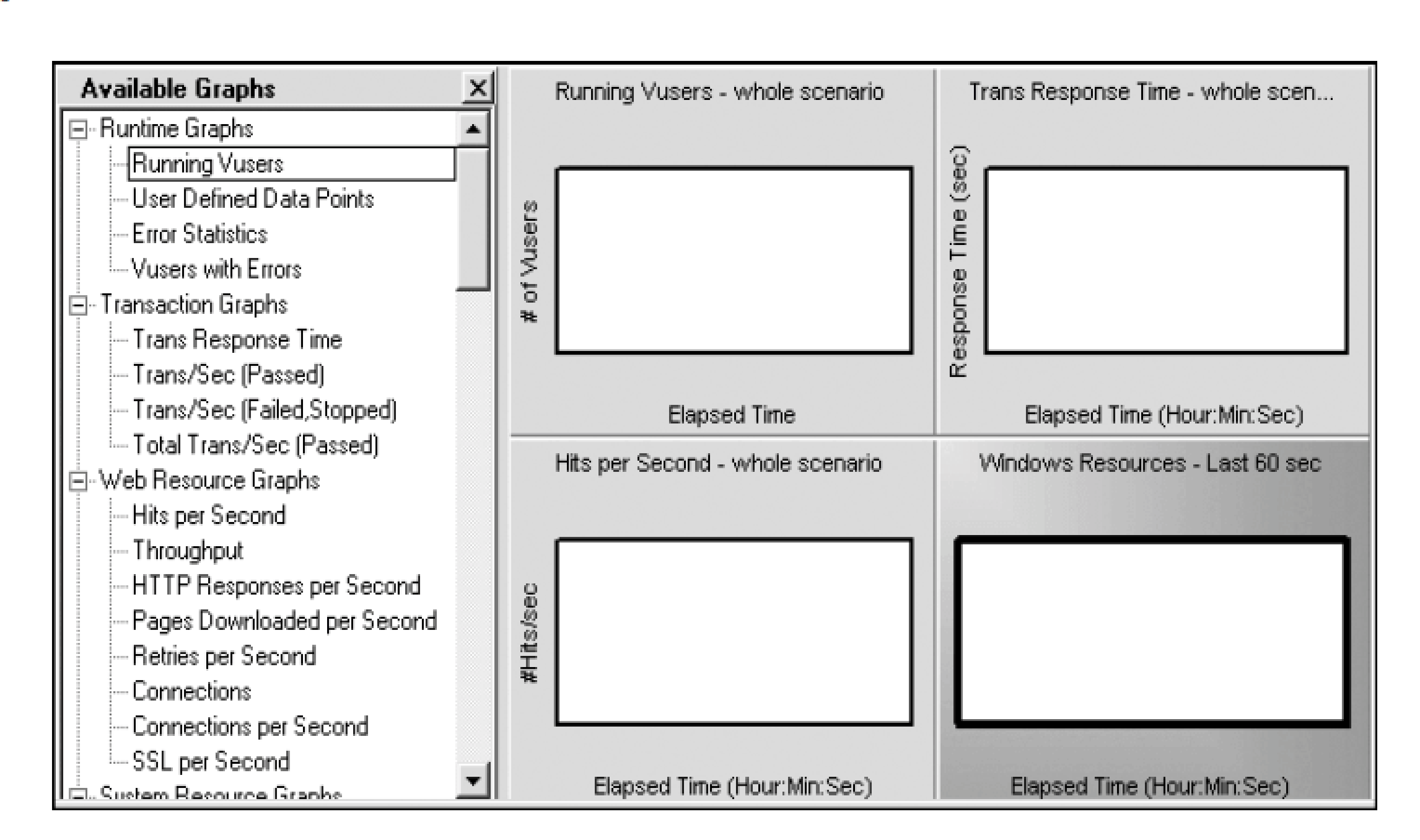


图 9.22　Run 视图

其中 Running Vusers 视图主要显示了特定时间内的虚拟用户数，Trans Response Time 视图显示了处理每一个业务的时间，Hits per Second 视图显示了每秒钟 HTTP 请求的次数，Windows Resources 显示了场景运行中 Windows 资源的情况。

9.3.5 运行场景并查看系统性能

添加负载并设置场景后，运行测试，这样就会为应用程序添加负载，并通过监视器观测应用程序在负载环境下的性能表现。

1. 运行负载测试

在 Controller Run 窗口中，单击 Start Scenario 按钮，在 Run 视图中可看到运行结果的视图显示，如图 9.23 所示。运行后的负载测试结果会以各种曲线的形式显示在各种视图中，要查看某条曲线的详细信息，可将鼠标放在曲线上，当鼠标变成小手形状时，单击鼠标，曲线变粗，在 Run 视图下方会显示各种颜色曲线的相关信息。

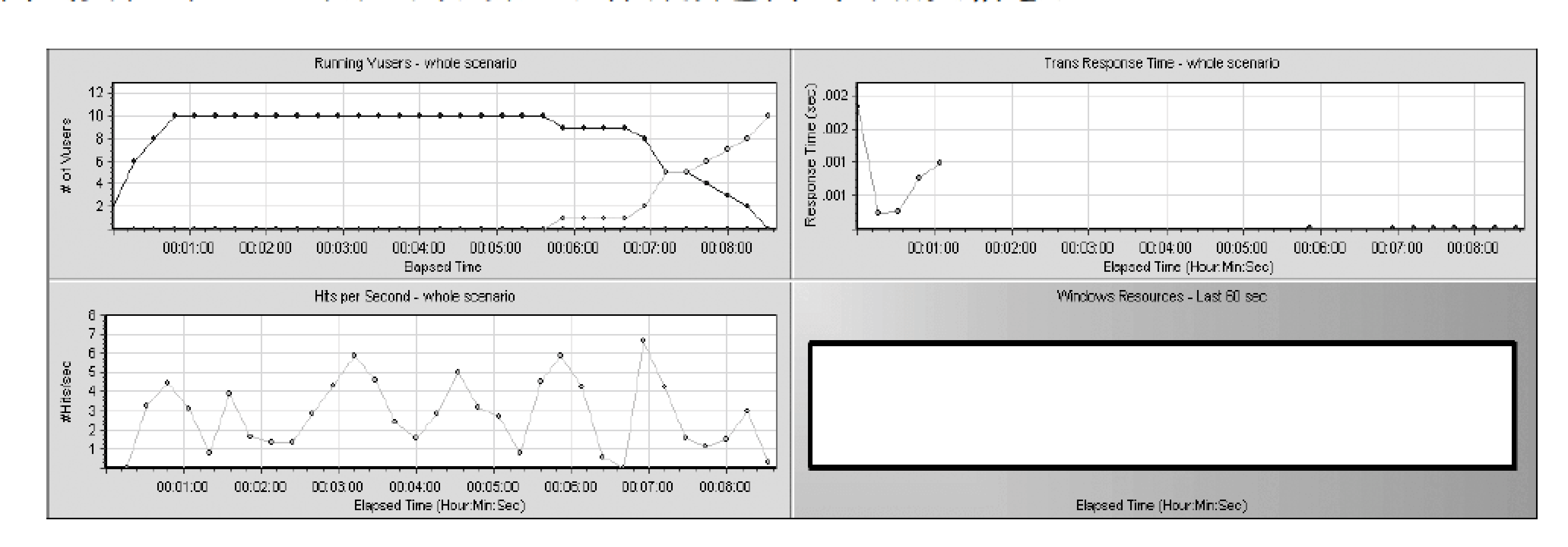


图 9.23 运行后的 Run 视图显示

2. 查看运行中的虚拟用户

负载测试中，可以实时查看运行场景中每个虚拟用户的动作，单击 Controller 窗口中的 Vusers...按钮，即可打开 Vusers 窗口，该窗口中可看到每个运行的虚拟用户，如图 9.24 所示。

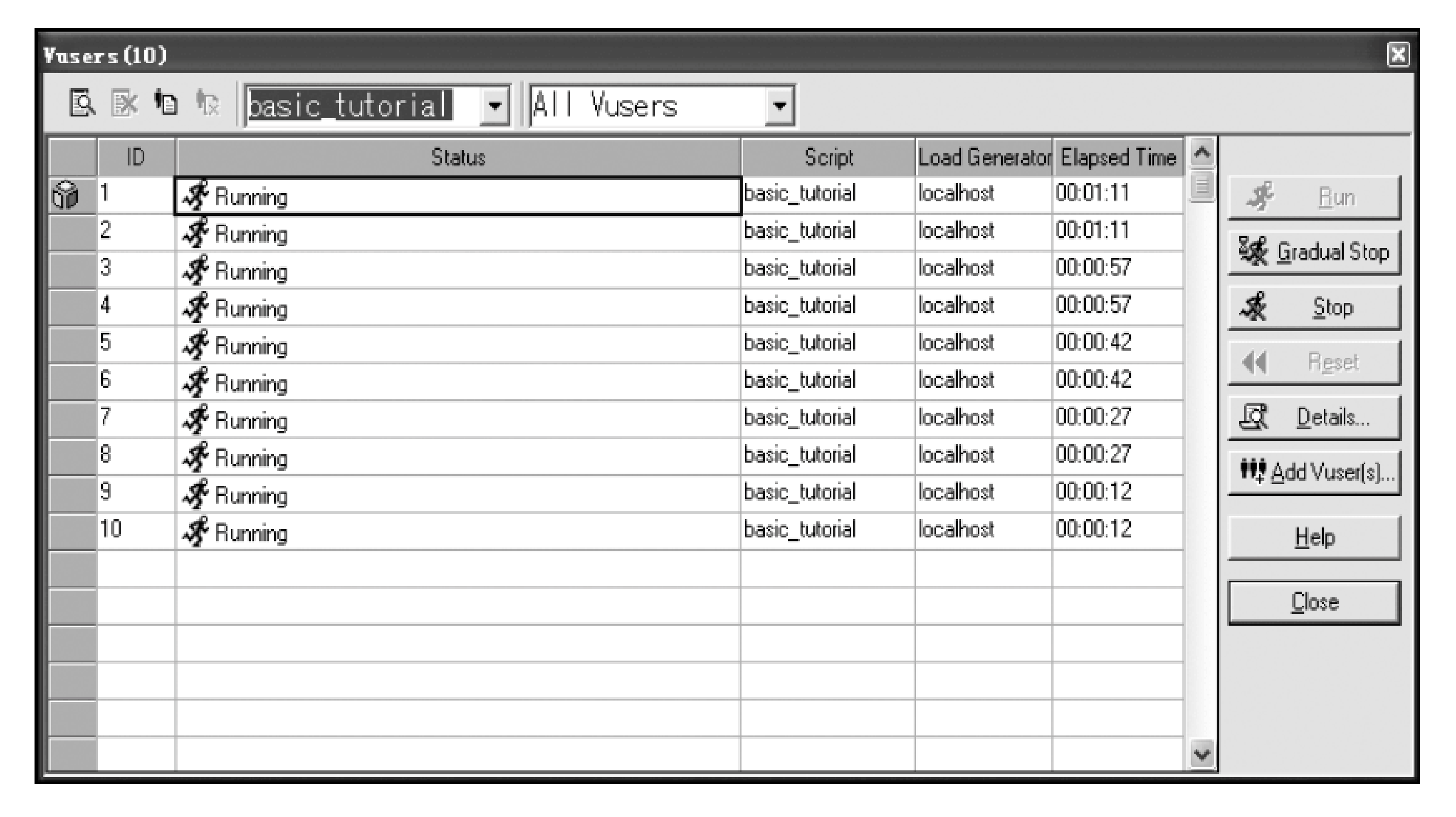


图 9.24 Vusers 窗口

要查看某个特定用户当前的动作，可选中虚拟用户列表中特定的用户，如用户 8，单击 Vusers 窗口左上角的第一个按钮——Show the selected Vusers 按钮，即可打开 Run-Time Viewer 窗口，如图 9.25 所示，即可看到虚拟用户 8 当前的动作。

或者单击 Vusers 窗口左上角第三个按钮，即 Show Vuser Log 按钮，则可打开 Vuser



图 9.25 Run-Time Viewer 窗口中查看虚拟用户运行时的动作

Log 窗口，如图 9.26 所示。该窗口中会显示虚拟用户执行过程中的动作。

3. 在测试中增加负载

在运行虚拟用户的过程中，也可以新增负载，如新增虚拟用户数等。通过单击 Controller 窗口中的 Run/Stop/Vusers...按钮，可打开 Run/Stop Vusers 窗口，如图 9.27 所示。如要增加 5 个虚拟用户，可在"#"列中输入 5，单击 Run 按钮，开始运行新的场景，并在 Controller Design 窗口中可看到虚拟用户数变成 15。

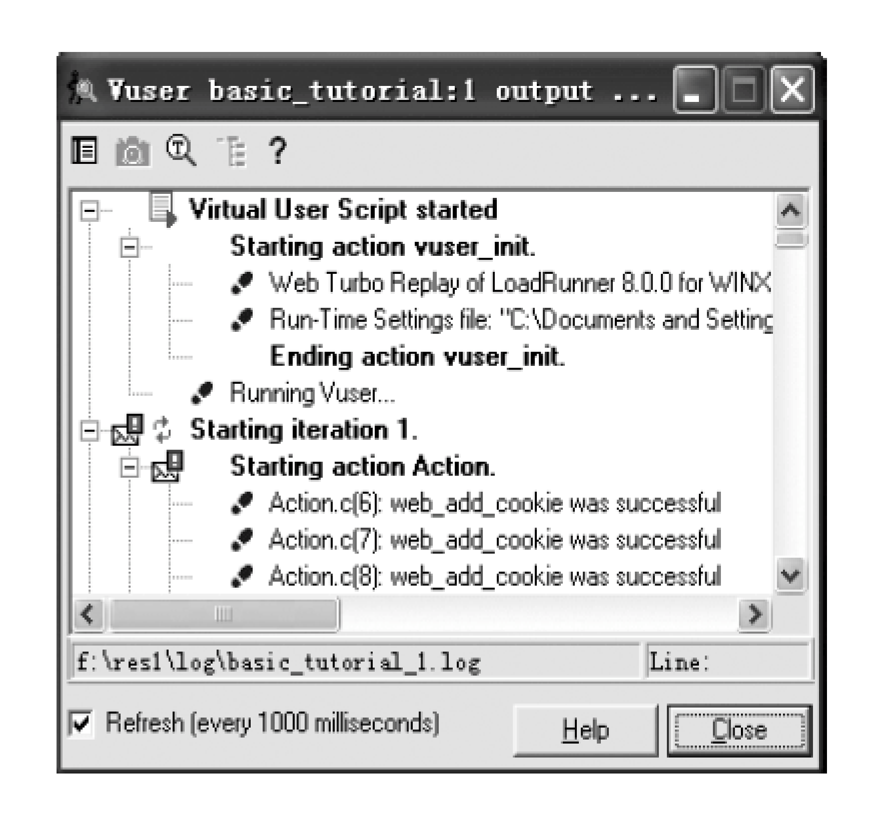


图 9.26 Vuser Log 窗口

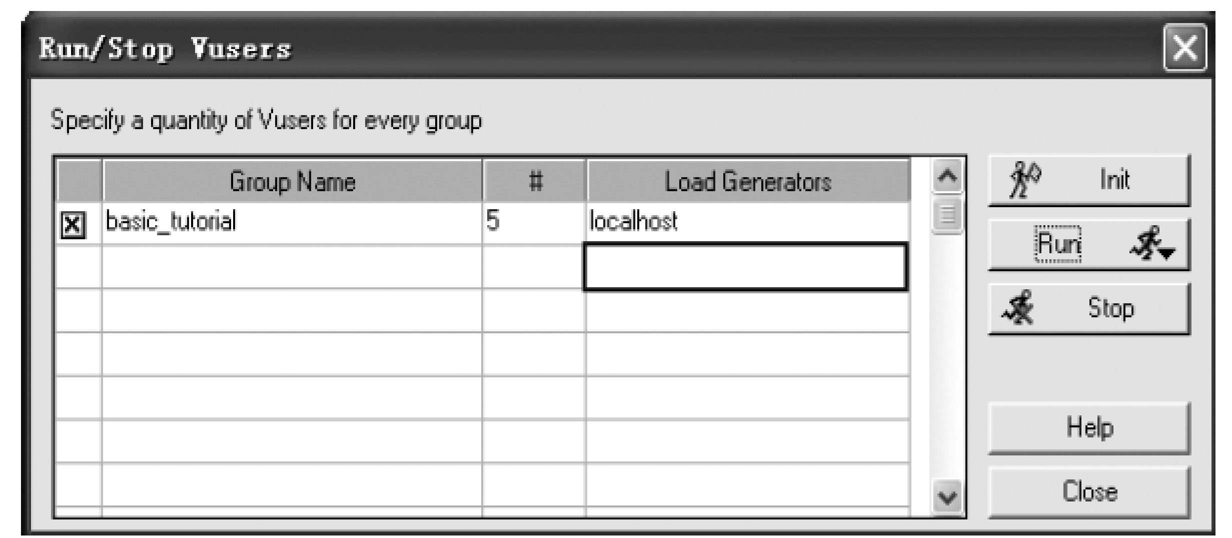


图 9.27 Run/Stop Vusers 窗口

4. 查看测试结果

场景运行结束之后，在 Controller 窗口右上角的结果统计窗口中，可显示运行的虚拟用户数、消耗的时间、通过测试的业务、失败的业务和出错等统计结果，如图 9.28 所示。

如要查看出错信息，单击 Errors 后面的数字，可打开 Output 窗口，如图 9.29 所示。可看到出错信息的描述，如要查看详细描述，可单击 Output 窗口中的 Details 按钮进行查看。

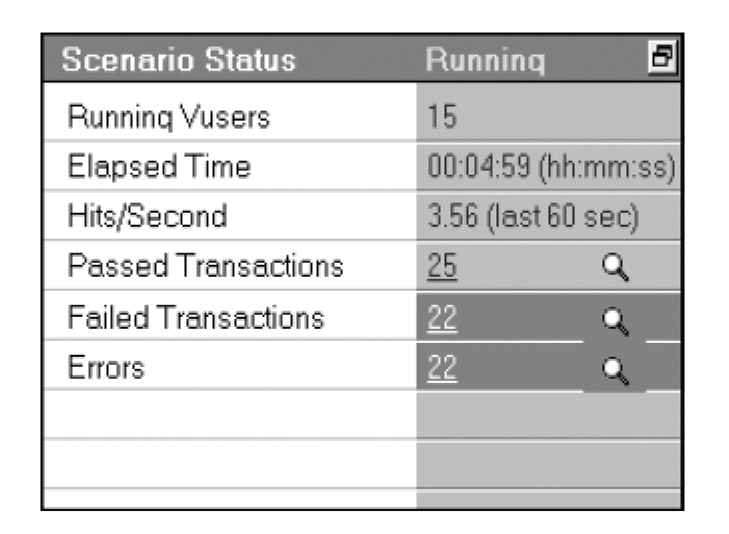


图 9.28 结果统计窗口

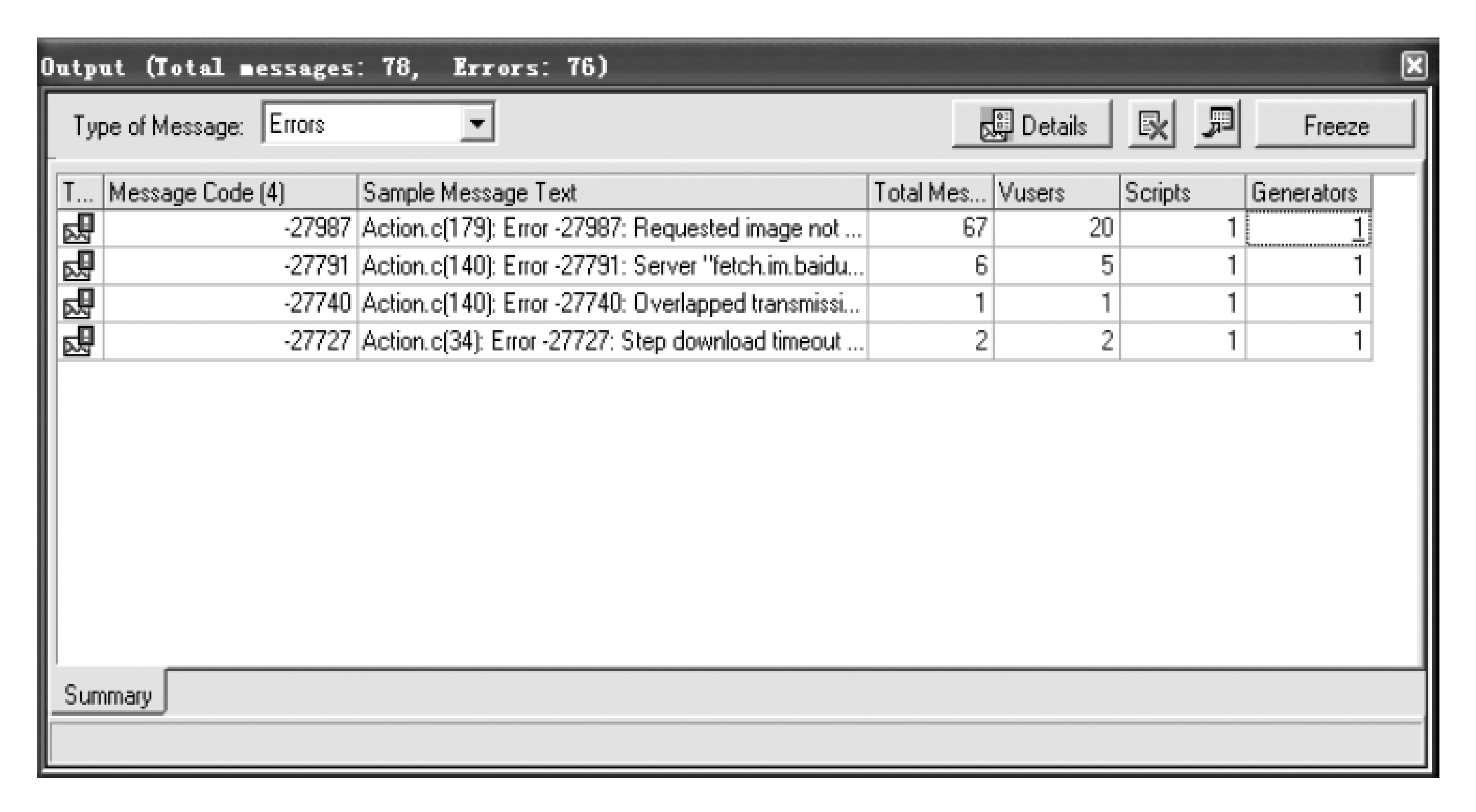


图 9.29　Output 窗口

9.3.6　结果分析

经过脚本录制、设计场景和运行场景之后，需要对运行结果进行分析，发现问题，以提高系统的性能。结果分析主要是依据负载测试期间 LoadRunner 工具生成的性能分析信息的图和报告。使用这些图和报告，可以标识和确定应用程序中的瓶颈，并确定需要对系统进行哪些更改来提高系统的性能。

Analysis Session 是脚本在场景运行后，对 LoadRunner 的运行结果文件(后缀为.lrr)进行修改后保存产生的 Session(会话)文件。Analysis Session 可以对 Session 进行保存，即可以保存对结果图进行处理过后的结果，这样不用每次打开测试结果都要按照以前的思路重新做一遍。不过 Analysis 默认是不自动保存 Session 的，如果要自动保存 Session，可在 Analysis 窗口中选择菜单 Tools→Templates→Apply/Edit Templates...，打开 Apply/Edit Template 对话框，如图 9.30 所示。选中 Automatically save the session as:复选框，并修改其后面要保存 Session 的路径，就可以自动保存 Session 到设定的路径中。如果每次运行场景后需要自动打开 Analysis，可在 Controller 窗口中选择菜单 Results→Auto Load Analysis，在每次场景运行结束后自动打开 Analysis，加载本次的返回结果。

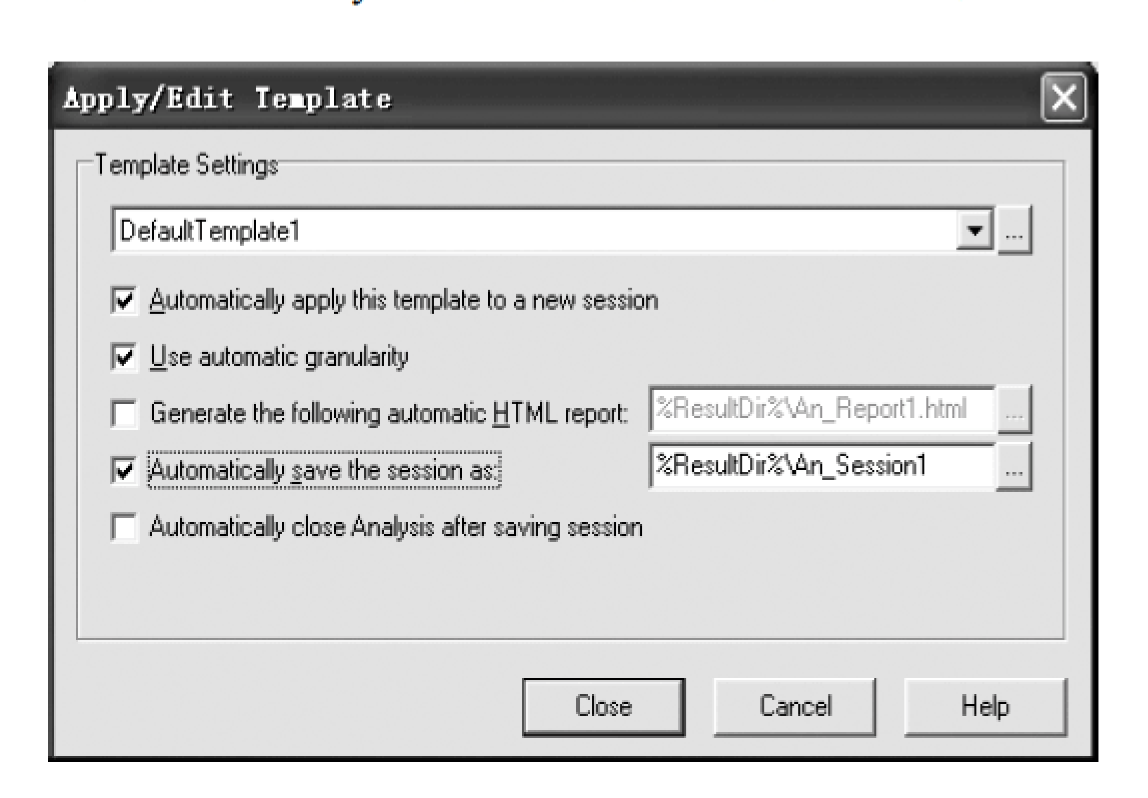


图 9.30　Apply/Edit Template 对话框

要进行结果分析，需要启动 LoadRunner 分析器，在图 9.5 中选择 Analyze Load Tests 按钮，则会打开 LoadRunner 分析器，如图 9.31 所示。

Analysis Session 文件的后缀为 *.lra，这里以系统中自带的一个 Session 为例，介绍结

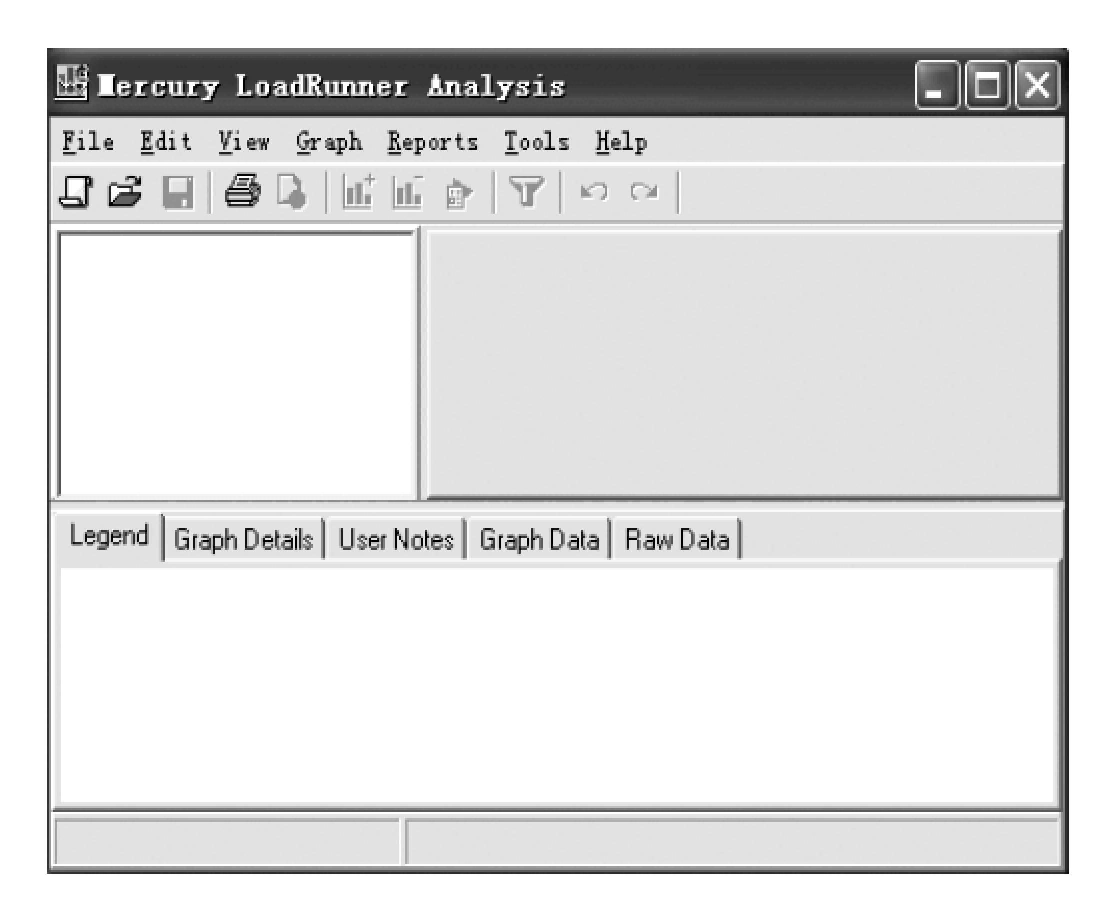


图 9.31　LoadRunner 分析器

果分析。

在 Analysis 窗口中，通过 File 菜单打开 LoadRunner 中的 tutorial 文件夹下的 analysis_session，如图 9.32 所示。可以看到 Transaction Summary 中的所有业务处理的响应时间，其中包括最小时间、最大时间、平均时间和 90%业务的响应时间等。

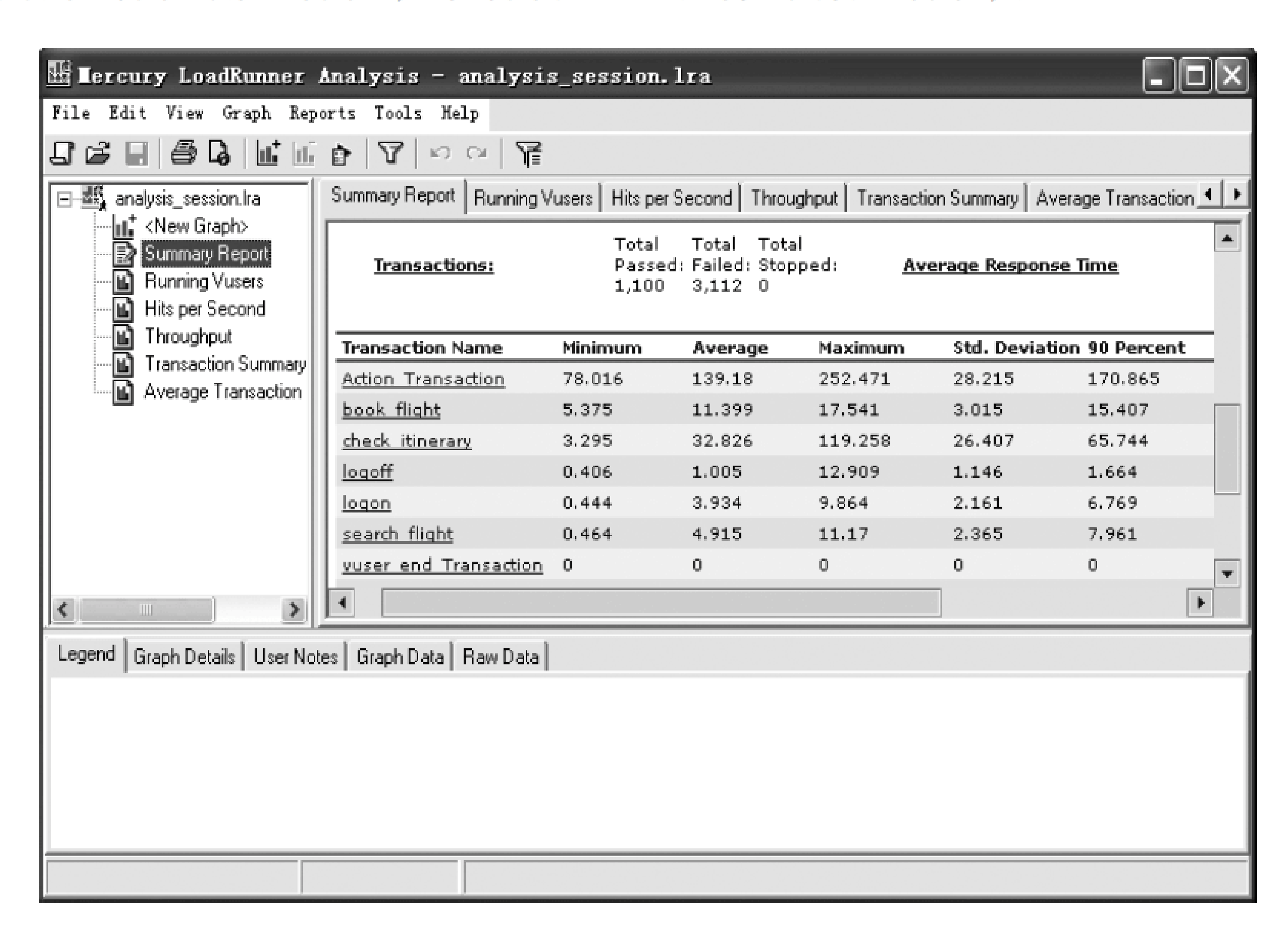


图 9.32　打开 analysis_session 的窗口

在这些业务中，check_itinerary 平均时间为 32.826 秒，而 90%的业务的响应时间为 65.744 秒，也就意味着这个业务大部分的操作响应时间高出平均响应时间的两倍，而且其失败次数也达到 28 次，因此该业务应该引起关注。单击打开 check_itinerary 业务，可以看到如图 9.33 所示的“平均业务响应时间”曲线图。图中用不同颜色曲线表示不同的业务，其中加粗的曲线为 check_itinerary 业务的响应时间曲线，每条曲线上的小方块表示特定时刻的业务响应时间，将鼠标放在小方块上就可以显示响应时间。可以看到，曲线图下方的多条曲线都比较平稳，也就意味着这些业务比较稳定，如 logon，logoff 等；而曲线图上方的两条曲线波动较大，即 Action_Transaction 和 check_itinerary 业务不稳定。check_itinerary 业务的响应时间在场景中的 2 分 56 秒时，响应时间达到最大 75.067 秒。

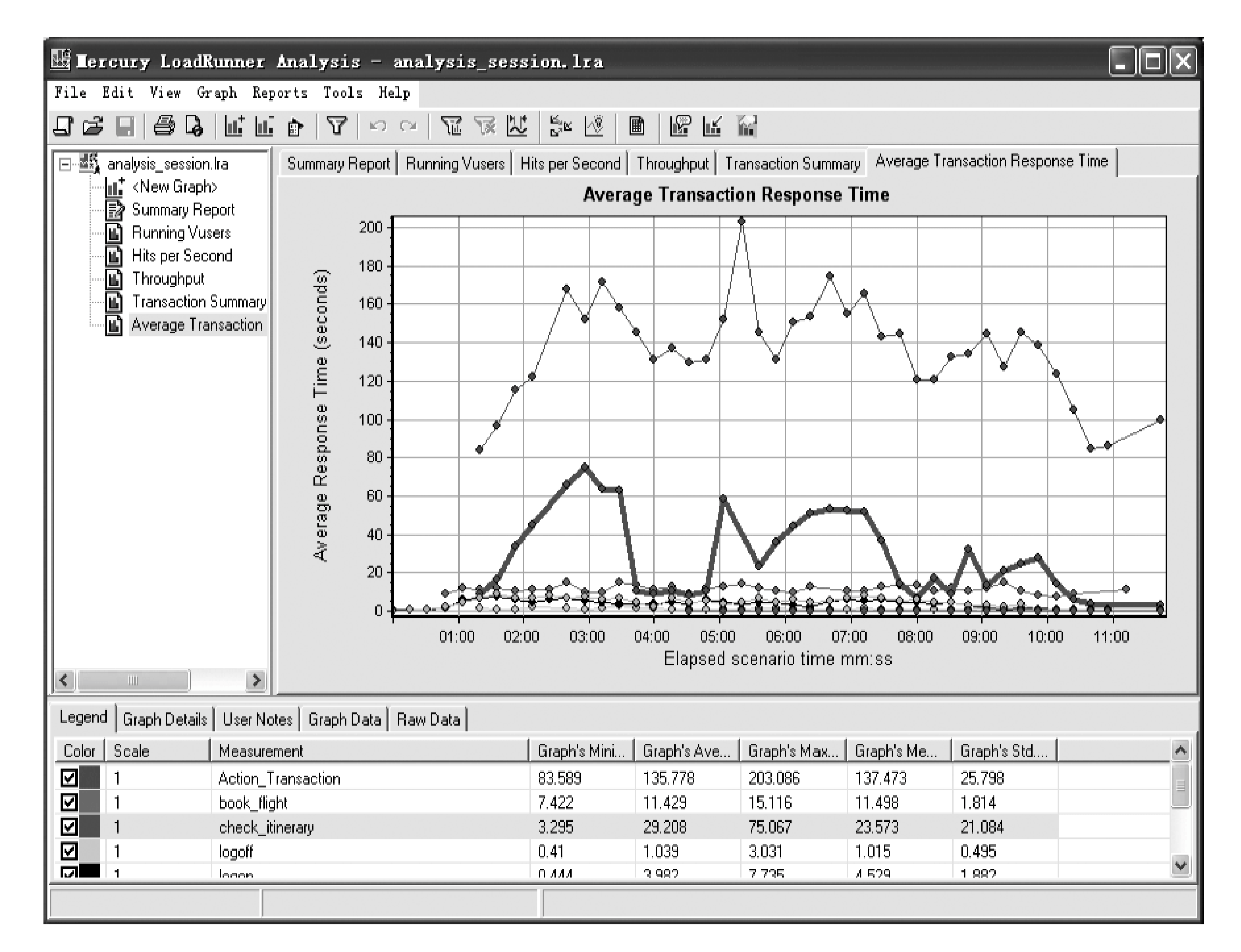


图 9.33 平均业务响应时间曲线图

在 LoadRunner 分析器中，还可以将多张图进行合并，如要观察平均响应时间随着虚拟用户数增加时的变化，可将 Running Vusers 和 Average Transaction Response Time 两张图进行合并。单击分析器上面的 Running Vusers 选项卡，打开 Running Vusers 图，如图 9.34 所示。可看到从开始到 1 分 30 秒之间虚拟用户数逐渐增加，从 1 分 30 秒到 4 分 30 秒之间 70 个虚拟用户同时运行，之后又逐渐减少。

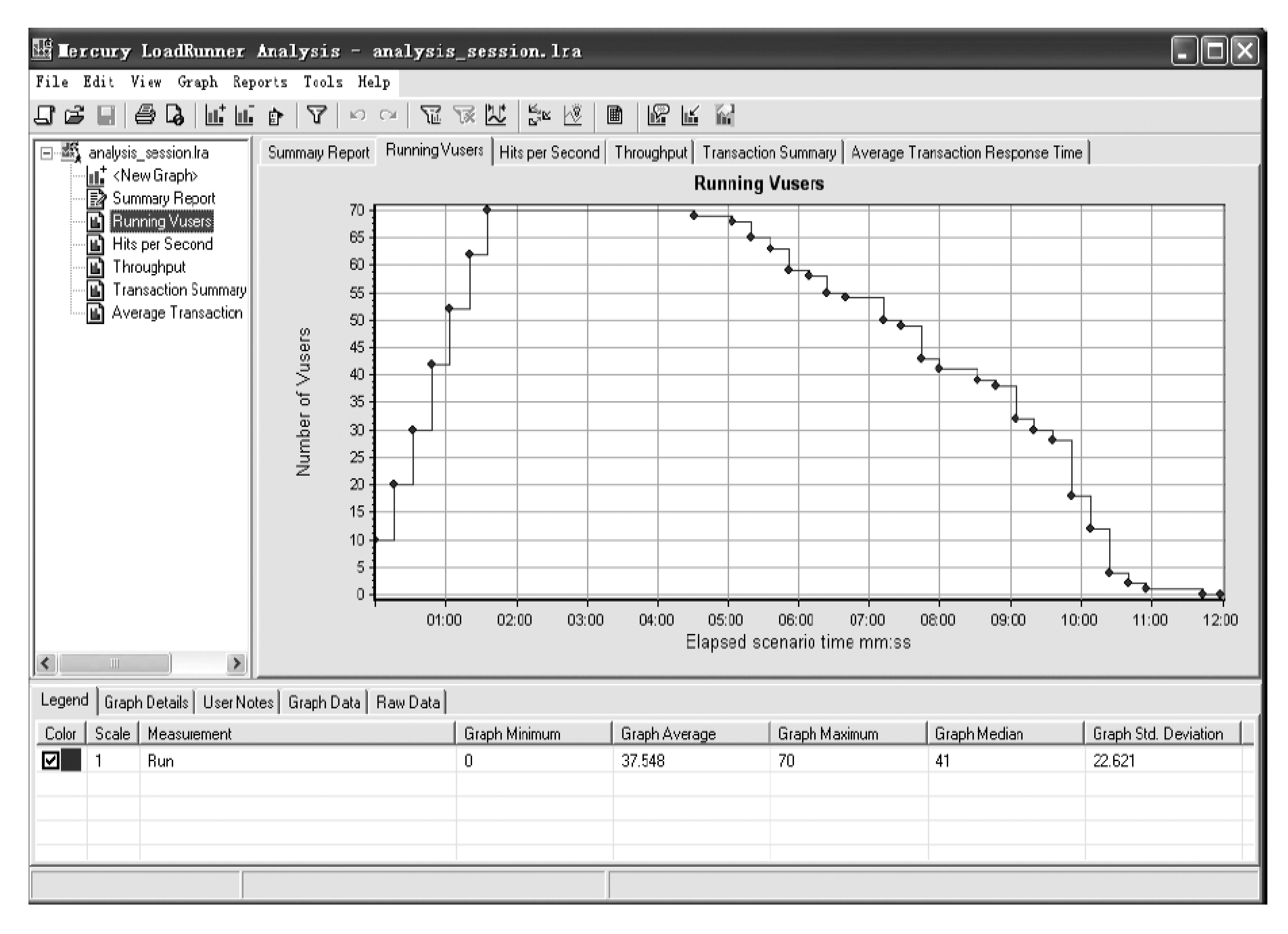


图 9.34 Running Vusers 分析图

若要查看70个虚拟用户同时运行时系统的响应时间，方法如下。

步骤一，在 Running Vusers 图形区域中右击，选择 Set Filter/Group By...菜单项，打开 Graph Settings 设置对话框，选择 Scenario Elapsed Time 选项后的下拉列表，设置要显示的开始时间和结束时间，如图9.35所示。

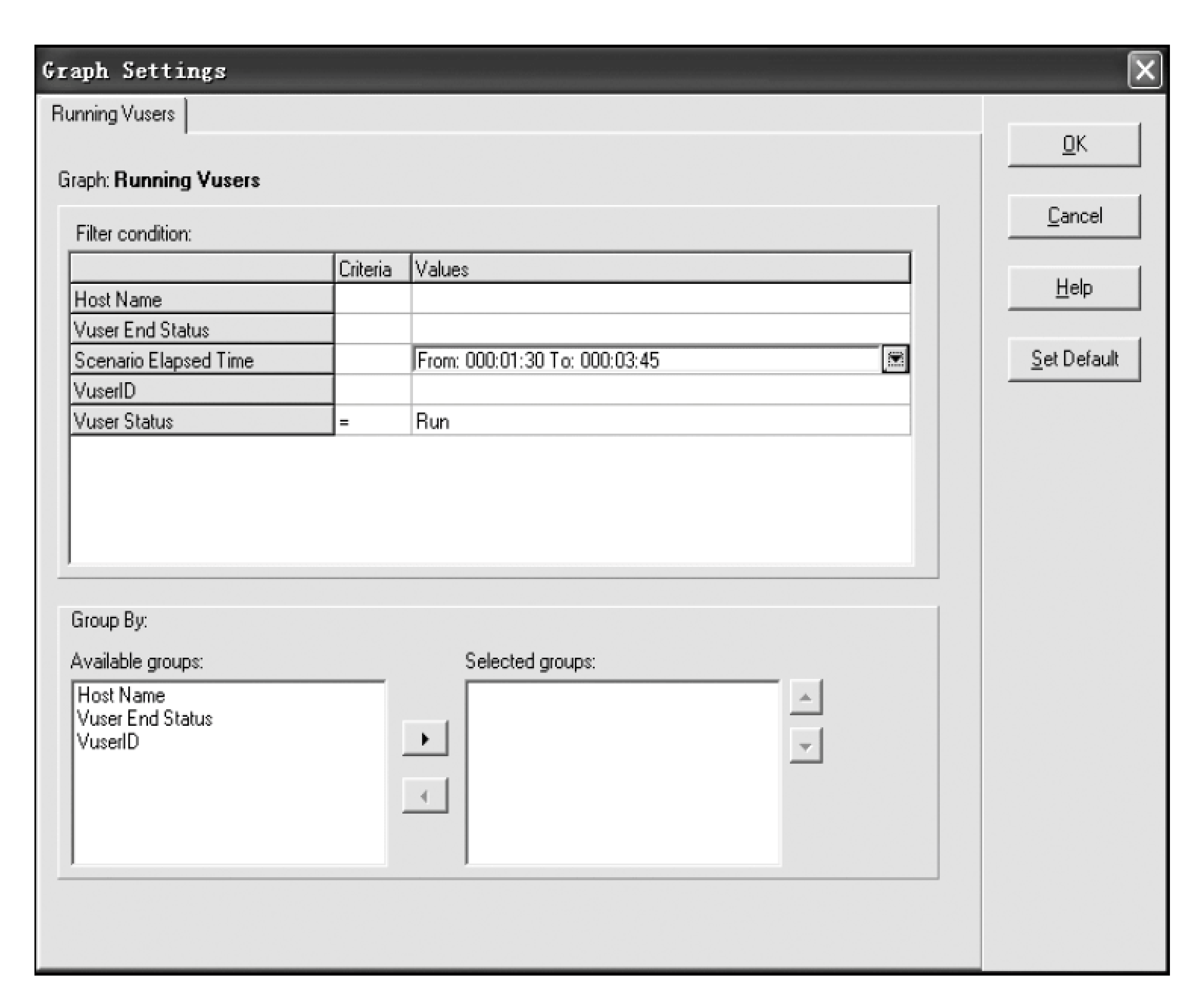


图9.35 Graph Settings 设置对话框

步骤二，在 Running Vusers 图形区域中右击，选择 Merge Graphs...菜单项，打开 Merge Graphs 对话框，如图9.36所示。在 Select graph to merge with 下面的下拉列表中选择 Average Transaction Response Time 选项，并选中 Select type of merge 下面的 Correlate 单选按钮，并单击 OK 按钮。

合并后的 Running Vusers-Average Transaction Response Time 合并图如图9.37所示。可以看到，当虚拟用户数为64时，系统的响应时间急剧增加。

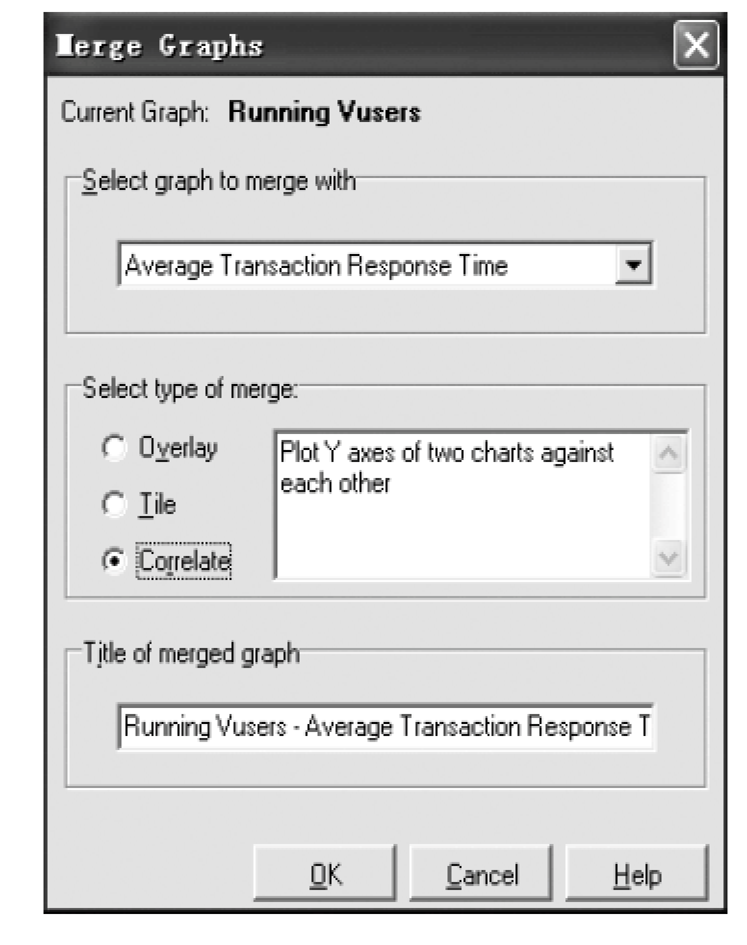


图9.36 Merge Graphs 对话框

9.3.7 分析影响性能的系统资源

从以上的分析可以看到，系统中的 check_itinerary 业务的响应时间会随着负载的增加而增加。下面来分析对系统产生影响的系统资源。

在图9.33所示的平均业务响应时间曲线图上右击，在快捷菜单中选择 Set Filter/Group By...菜单，打开 Graph Settings 对话框，如图9.38所示。

在 Filter condition 中的 Transaction Name 后的 Values 中选择 check_itinerary 选项，并单击 OK 按钮。平均业务响应时间曲线图会单独显示 check_itinerary 业务响应时间，如

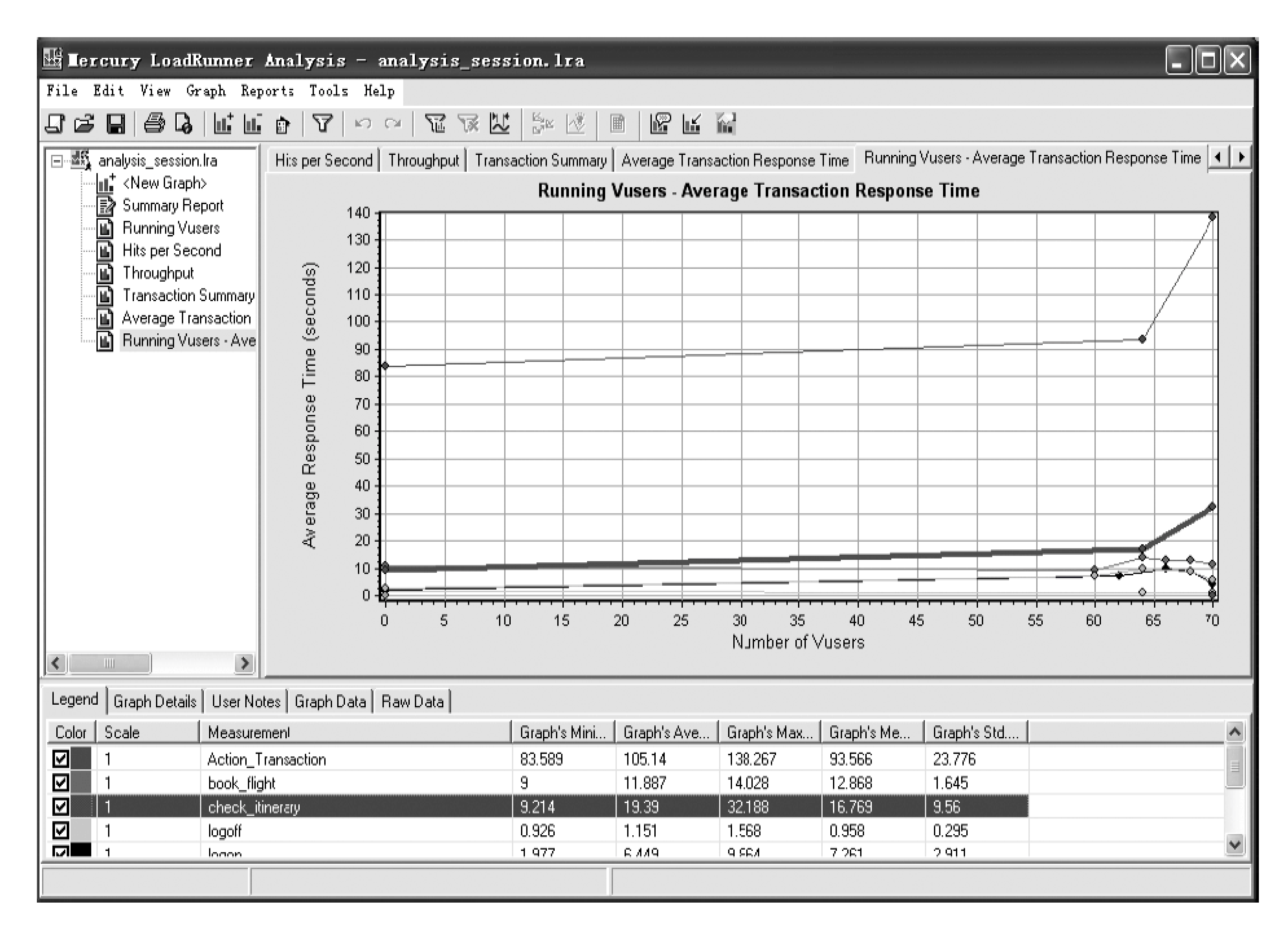


图 9.37　Running Vusers-Average Transaction Response Time 合并图

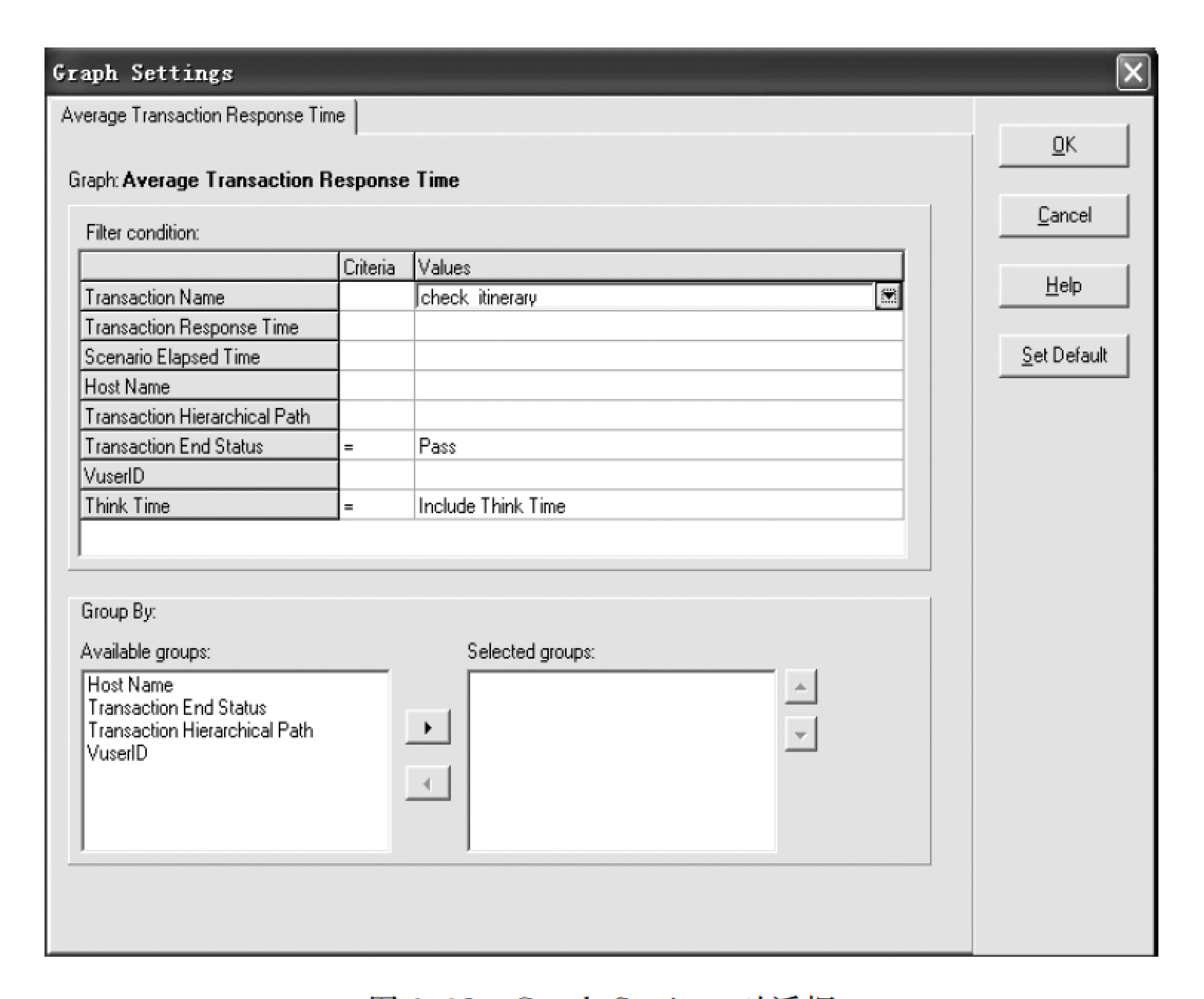


图 9.38　Graph Settings 对话框

图 9.39 所示。

在图 9.39 的响应时间曲线图上右击，在快捷菜单中选择 Auto Correlate...菜单，打开 Auto Correlate 对话框，如图 9.40 所示。Auto Correlate 可以将对某个特定业务响应时间有影响的所有图进行合并。拖动图中红色或绿色柱，选择时间范围，如曲线图下方的时间范围为 From 000:01:20 To 000:03:40，单击 OK 按钮，分析器中的曲线图如图 9.41 所示。该

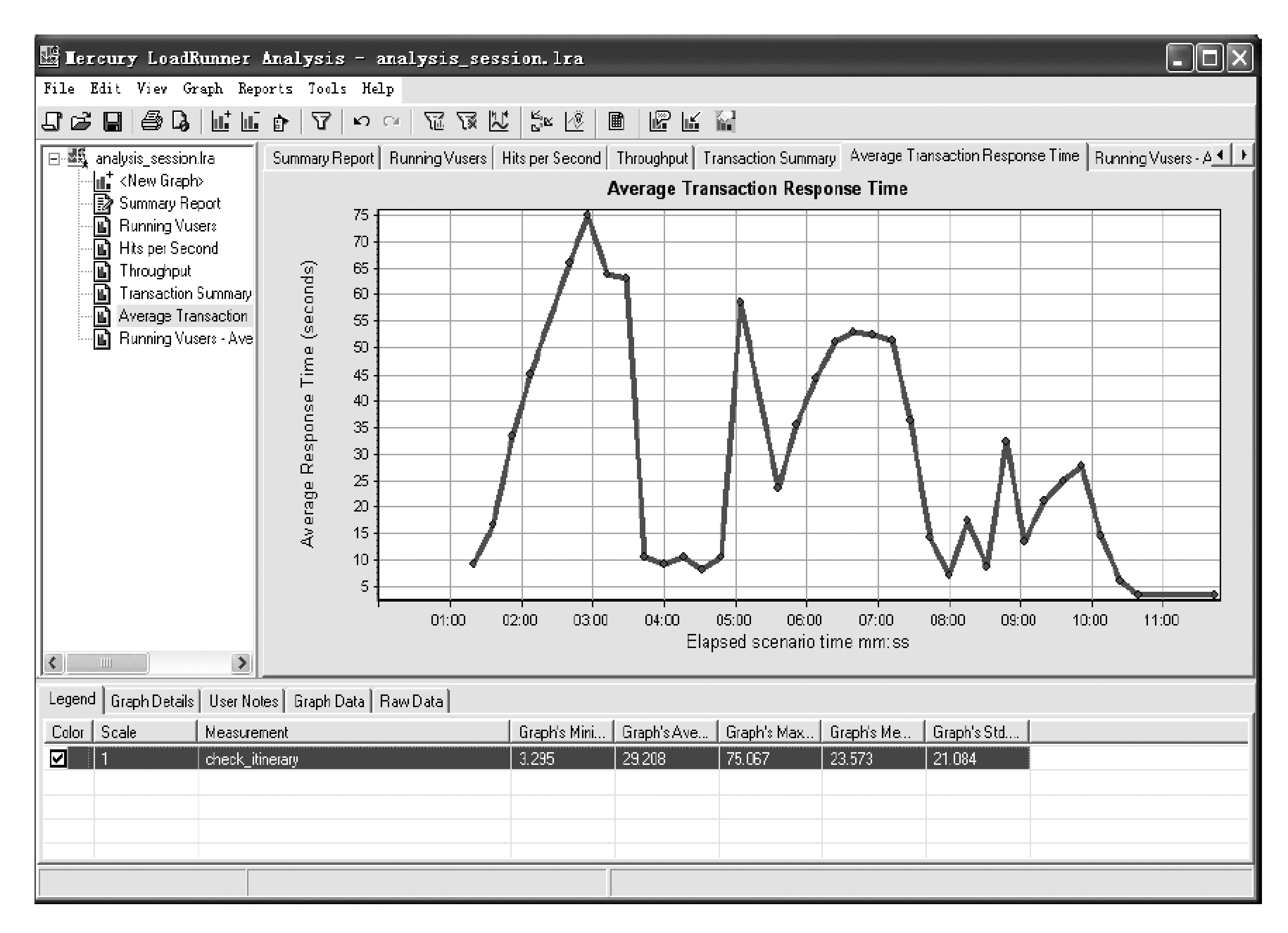


图 9.39　过滤后的 check_itinerary 平均响应时间曲线图

图中显示了所有对系统性能有影响的资源。其中在曲线图下方的 legend 选项窗口中，前两个资源，即列 Measurement 中为 Private Bytes 和 Pool Nonpaged Bytes 的资源的 Correlation Match 分别为 76 和 75，其他资源都为 50 以下，即这两个资源对该业务的响应时间影响最大。也就是当该业务的响应时间达到峰值时，系统资源将会短缺。

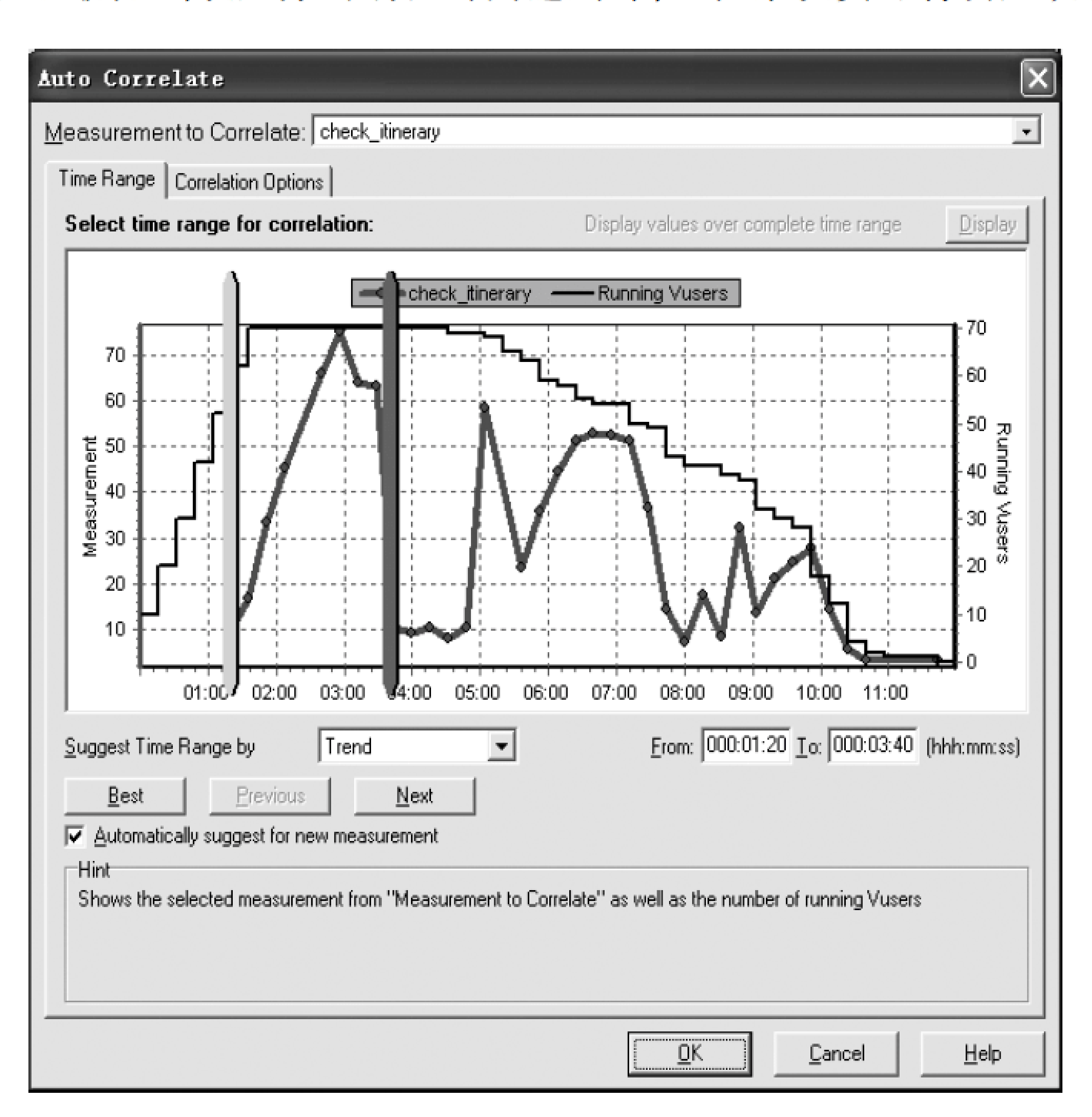


图 9.40　Auto Correlate 对话框

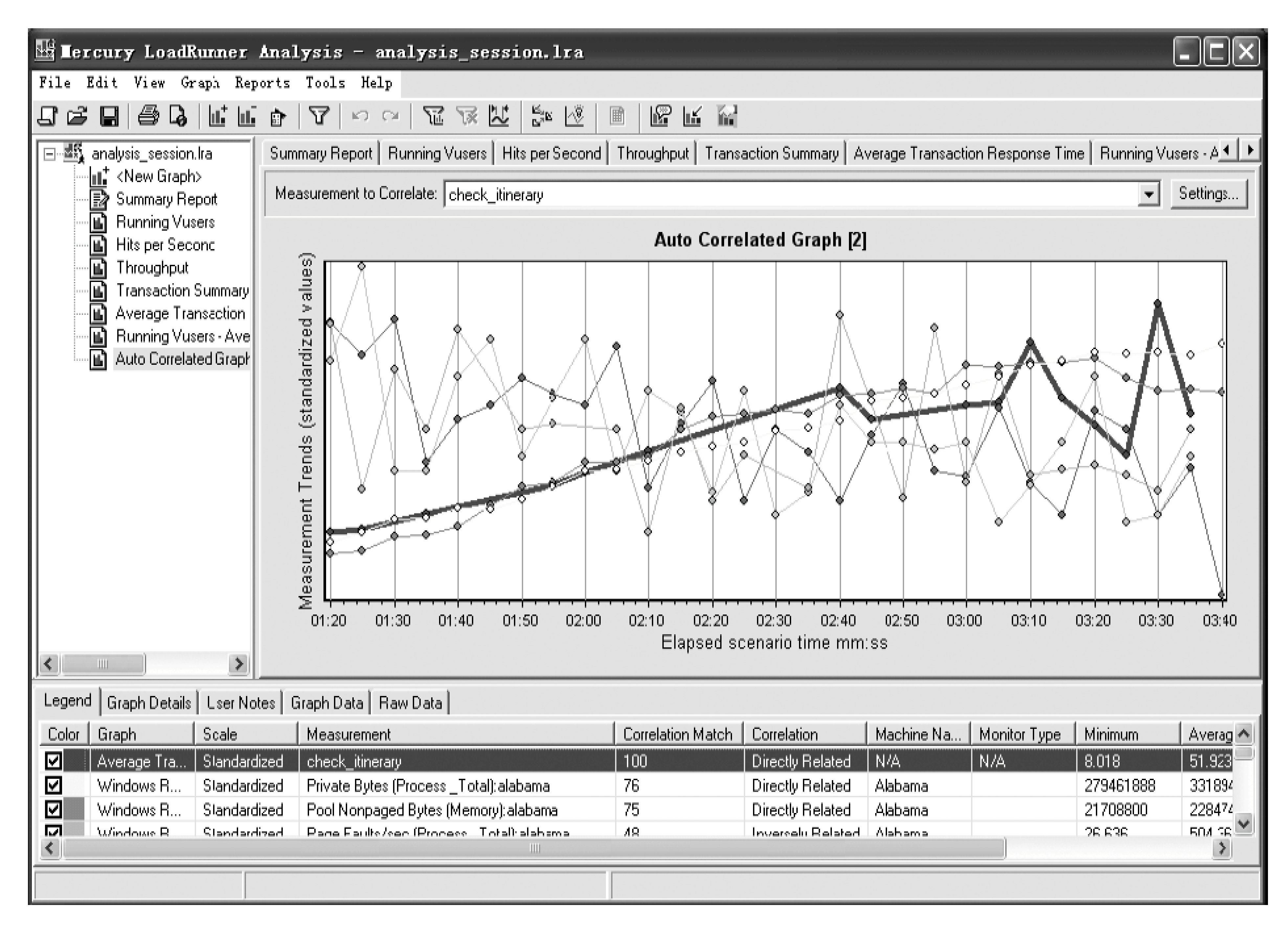


图 9.41 自动关联后的曲线图

9.3.8 发布性能测试结果

用户可以将测试的结果以 HTML 或 Word 文档形式发布测试报告,只需在分析器中的 Reports 下选择 HTML Report…或者 Microsoft Word Report…,即可发布测试报告。如图 9.42 所示,是以 HTML 形式发布的测试报告。

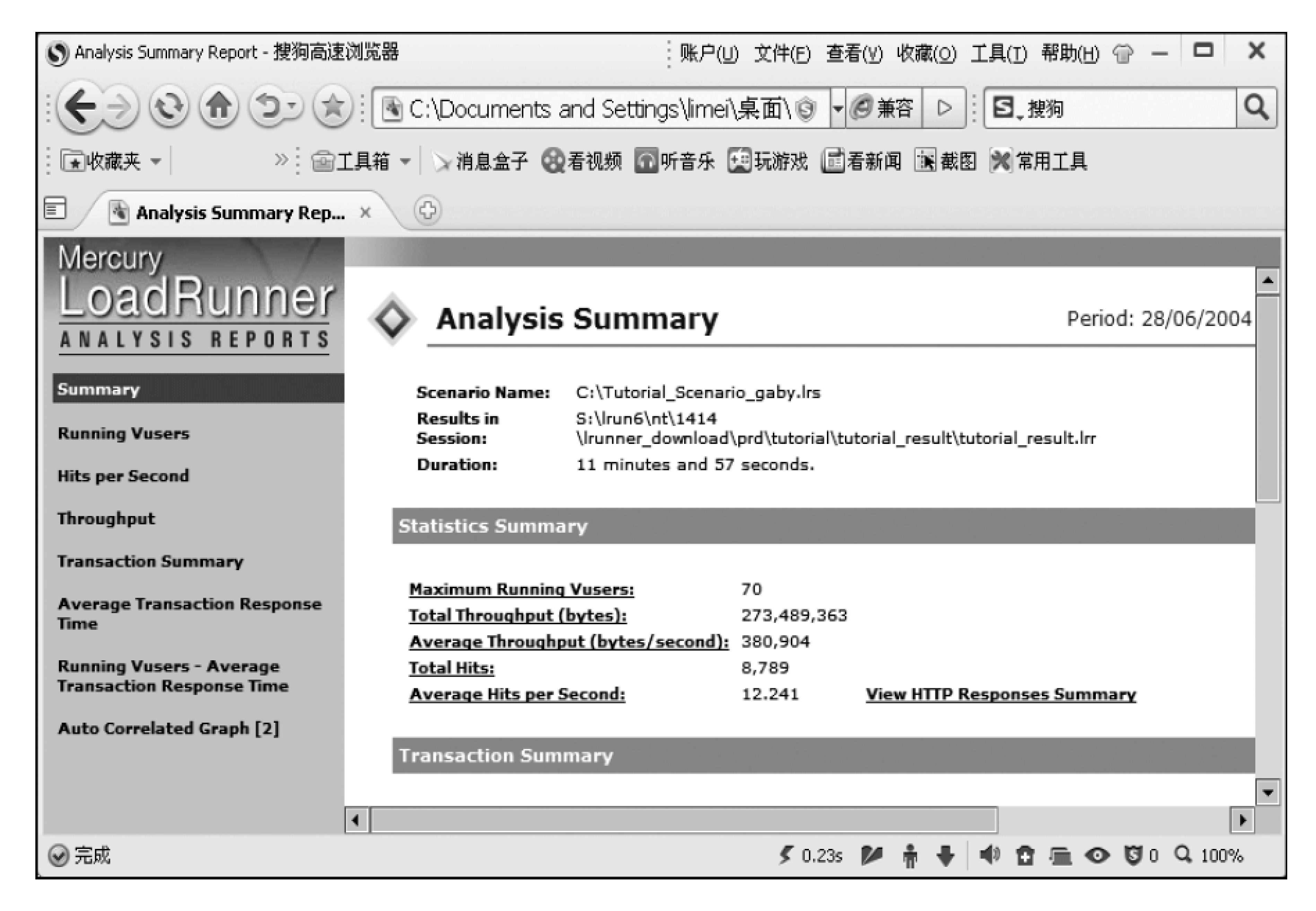


图 9.42 HTML 形式的测试报告

9.4 本章小结

本章主要介绍了性能测试，首先介绍了性能测试的目的和性能测试前的准备，接着就性能测试工具及相关测试网站作了分类介绍，最后主要介绍了性能测试工具 LoadRunner，以 LoadRunner 8.0 为例分别对 LoadRunner 的 6 个测试步骤——测试计划，创建脚本，设计场景，运行场景，监视场景和分析结果进行详细介绍。每个测试步骤均由一个 Mercury LoadRunner 工具组件执行，其中包括 Mercury 虚拟用户生成器（VuGen）、Mercury LoadRunner Controller 和 Mercury Analysis。

习　题

1. 判断对错。

(a) LoadRunner 进行性能测试时，需要先准备脚本文件和场景文件。

(b) LoadRunner 使用虚拟用户 Vuser 模拟真实用户操作被测试系统。

(c) LoadRunner 在回放脚本时，各步骤间的思考时间按照录制时确定，不可以发生改变。

(d) LoadRunner 脚本使用 C 语言编写。

(e) 如果要能够查看到 Vuser 运行时候的 log，必须在 Run-time Settings 对话框中启动日志。

(f) LoadRunner 可以在运行场景的过程中增加 Vuser 数量。

2. 请列举出常用的性能测试工具。

3. LoadRunner 由哪些部分组成？

4. 简述利用 LoadRunner 进行性能测试的过程。

5. 采用面向对象的方法实现 NextDate 问题，并利用 LoadRunner 进行性能测试。

第 10 章　IBM Rational ClearQuest 缺陷跟踪管理

缺陷是指程序中存在的错误，如语法错误、拼写错误等。缺陷可能出现在设计中，甚至在需求、规格说明或其他的文档中。软件缺陷包括文档缺陷、代码缺陷、测试缺陷和过程缺陷。软件缺陷可以导致系统不能实现其功能，引起系统的失效。

软件缺陷具有单一准确性，所以每个报告只针对一个软件缺陷。在一个报告中报告多个软件缺陷常常会使部分缺陷未被注意和修复，最终导致软件不能被彻底修正。

缺陷可以再现，报告中提供缺陷的精确操作步骤，在回归测试中，让测试人员再现这个缺陷。测试报告中需要提供完整、前后统一的软件缺陷的步骤和信息，如图片信息和 log 文件等，通过使用关键词，使软件缺陷的标题短小简练，准确解释产生缺陷的现象，如“主页”、“导航栏”、“分辨率”等关键词。另外，需要在报告中指明测试的特定条件，以及将缺陷的内容补充完整。最后，在进行缺陷描述时，不能带感情色彩，不能做任何评价。

缺陷的具体定义为：

- 软件没有达到产品说明书的功能。
- 程序中存在语法错误。
- 程序中存在拼写错误。
- 程序中存在不正确的程序语句。
- 软件出现了与产品说明书不一致的表现。
- 软件功能超出产品说明书的范围。
- 软件没有达到用户期望的目标。
- 测试员或用户认为软件的易用性差。

按照定义，将缺陷分为文档缺陷、代码缺陷、测试缺陷和过程缺陷。

(1) 文档缺陷是指在对文档通过测试需求分析、文档审查进行静态检查的过程中发现的缺陷。

(2) 代码缺陷是指对代码进行同行评审、审计或代码走查过程中发现的缺陷。

(3) 测试缺陷是指由测试活动发现的被测对象(被测对象一般是指可运行的代码和系统，不包括静态测试发现的问题)的缺陷。测试活动主要包括内部测试、连接测试、系统集成测试和用户验收测试。

(4) 过程缺陷是指通过过程审计、过程分析、管理评审、质量评估和质量审核等活动发现的关于过程的缺陷和问题。过程缺陷的发现者一般是质量经理、测试经理和管理人员。

按阶段分，软件缺陷分为需求缺陷、设计缺陷、结构缺陷、系统缺陷、系统结构缺陷、测试设计与测试执行错误；按涉及的方面不同，软件缺陷可以分为功能类、性能类、系统/模块接口类、用户界面类、数据处理类、流程类、提示信息类、软件包类、建议类、常识类和文档类，如表 10.1 所示。

表 10.1 软件缺陷类型表

缺陷类型	描述
需求缺陷	需求有误;需求逻辑错误;需求不完备;需求文档描述有问题;需求更改
设计缺陷	功能的使用给用户带来不便及不符合行业标准的;设计不合理;设计文档描述有问题;设计变更带来的问题
结构缺陷	控制流程和控制顺序错误;处理错误
系统结构缺陷	操作系统引用或使用错误;软件结构错误;恢复错误;执行错误;诊断错误;分割覆盖错误;引用环境错误
测试设计与测试执行错误	测试设计错误;测试执行错误;测试文档错误;测试用例不充分;其他测试错误
功能类	影响了各种系统功能逻辑的缺陷;重复的功能;多余的功能;功能实现与设计要求不符;功能使用性、方便性、易用性不够
性能类	不满足系统可测量的属性值;事务处理速率、并发量、数据量、压缩率和响应的时间不符合系统设计指标
系统/模块接口类	与其他组件、模块或设备驱动程序的接口冲突;调用参数、控制块或参数列表等不匹配,发生冲突
用户界面类	影响了用户界面、人机交互特性;屏幕格式、用户输入灵活性、结果输出格式等方面的缺陷;界面不美观;控制排列、格式不统一;焦点控制不合理或不全面
数据处理类	数据有效性检测不合理;数据来源不正确;数据处理过程不正确
流程类	流程控制不符合要求;流程实现不完整
提示信息类	提示信息重复或出现时机不合理;提示信息格式不符合要求;提示框返回后焦点停留位置不合理
软件包类	由软件配置库、变更管理或版本控制引起的错误
建议类	功能性建议;操作建议;校验建议;说明建议
常识类	违背正常习俗习惯的,比如日期、节日等
文档类	影响发布和维护,包括注释、用户手册和设计文档

缺陷状态是指缺陷通过一个跟踪修复过程的进展情况。缺陷状态分为以下几个。

(1) 激活或打开:问题还没有解决,存在于源代码中,确认"提交的缺陷",等待处理,如新报的缺陷。

(2) 已提交:测试人员发现缺陷后提交到缺陷管理系统中的状态(初始状态)。

(3) 已修改:已经被程序员检查,修复过的缺陷,通过单元测试,认为已经解决,但还没有被测试人员验证提交到缺陷管理系统中的状态。

(4) 不修改(保留):程序员或项目经理根据需求分析、概要设计和详细设计说明书等的要求,经过考虑后决定对缺陷不进行修改(由于技术原因或第三者软件的缺陷,开发人员不能修复的缺陷),其缺陷的状态为不修改。

(5) 延迟:根据目前项目进程或计划等情况,暂时延期的状态,缺陷可以在下一个版本中解决。

(6) 待讨论:讨论后才能决定是否需要修改的缺陷的状态。

(7) 已验证:已经解决的并通过测试人员复测的缺陷的状态。

(8) 关闭：问题完全解决了，只供以后备查的状态。

(9) 重新打开：测试人员验证后，依然存在缺陷，等待开发人员进一步修复，重新打开以前关闭的缺陷状态。

在 Bug 工具中，可以自己定制适合项目的状态项，比如废除、拒绝等。

缺陷管理工具很多，目前使用比较多的是 ALM 以及 Rational 系列的 ClearQuest。本书接下来继续介绍 IBM Rational ClearQuest 的使用。Rational ClearQuest 是 IBM 公司一款完整的缺陷跟踪和变更管理工具，它的功能强大、灵活，可实现流程自定义、查询自定义以及用户权限分级管理等功能，并可以集成 Crystal Report（水晶报表），实现更加灵活的报表自定义功能。ClearQuest 可以与 Rational 系列的其他产品结合使用。

10.1 工具安装及基本使用

Rational ClearQuest 可以安装在 Windows NT 4.0/2003/2000/XP 上。ClearQuest 的安装十分简单，双击 setup.exe 文件，选择 Rational ClearQuest，一路选择"下一步"就可以了，如图 10.1 至图 10.6 所示。



图 10.1 安装图解 1

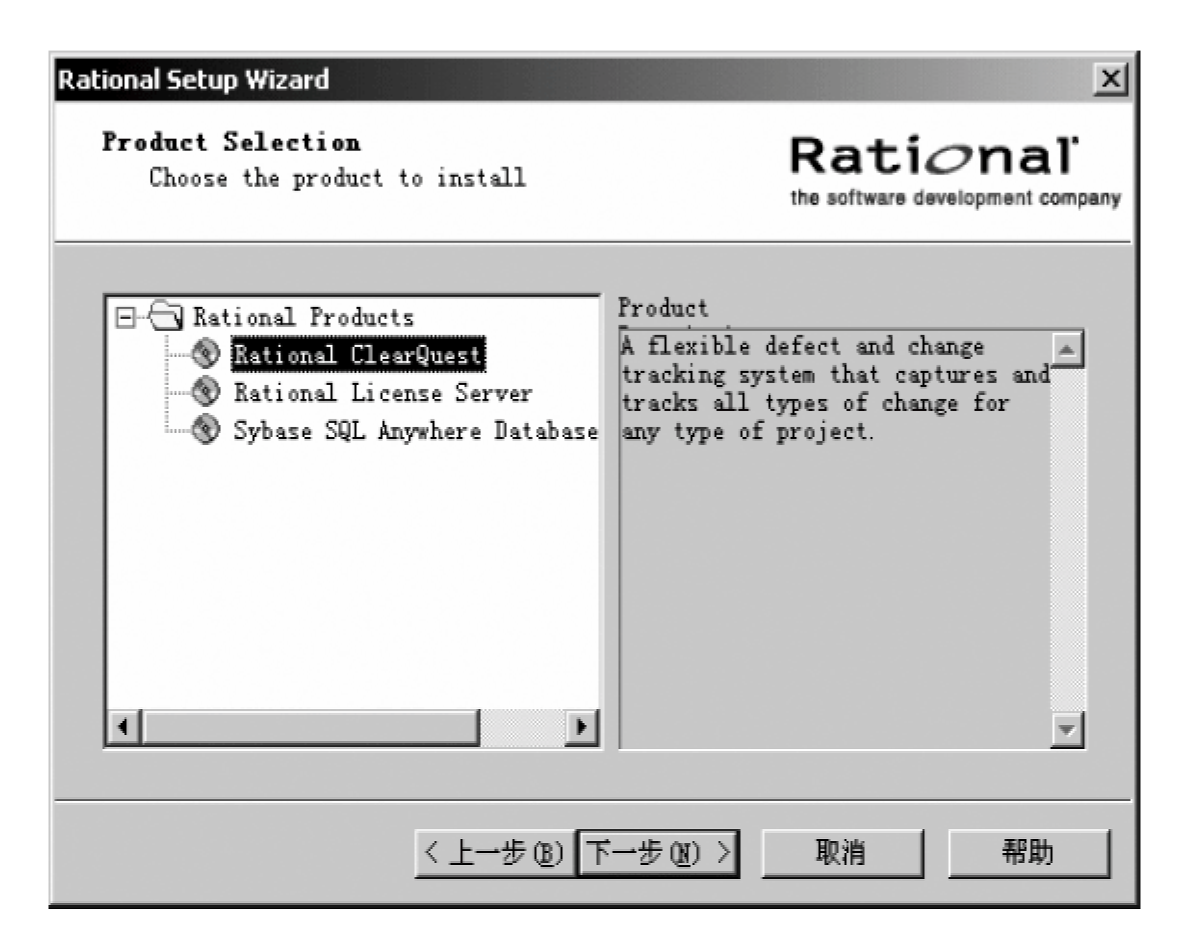


图 10.2 安装图解 2

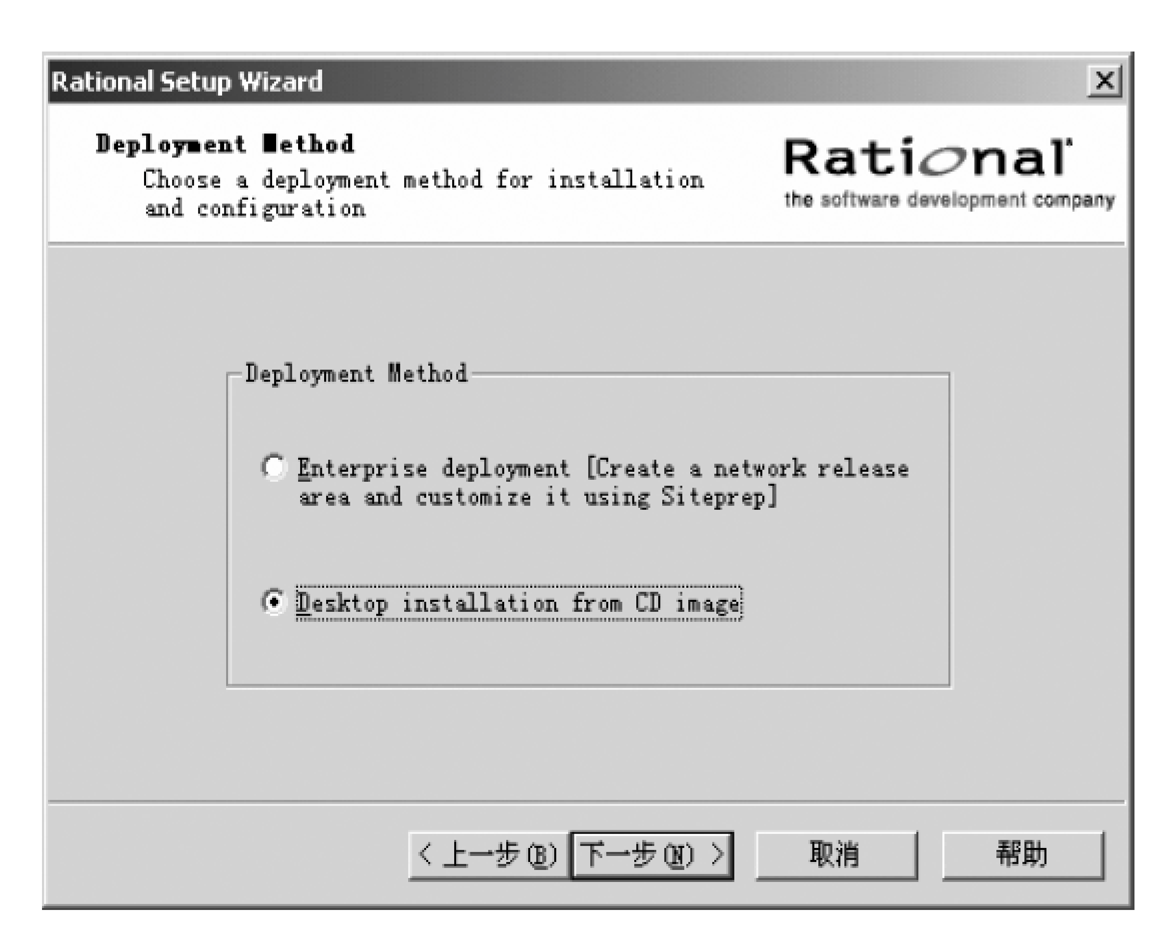


图 10.3　安装图解 3

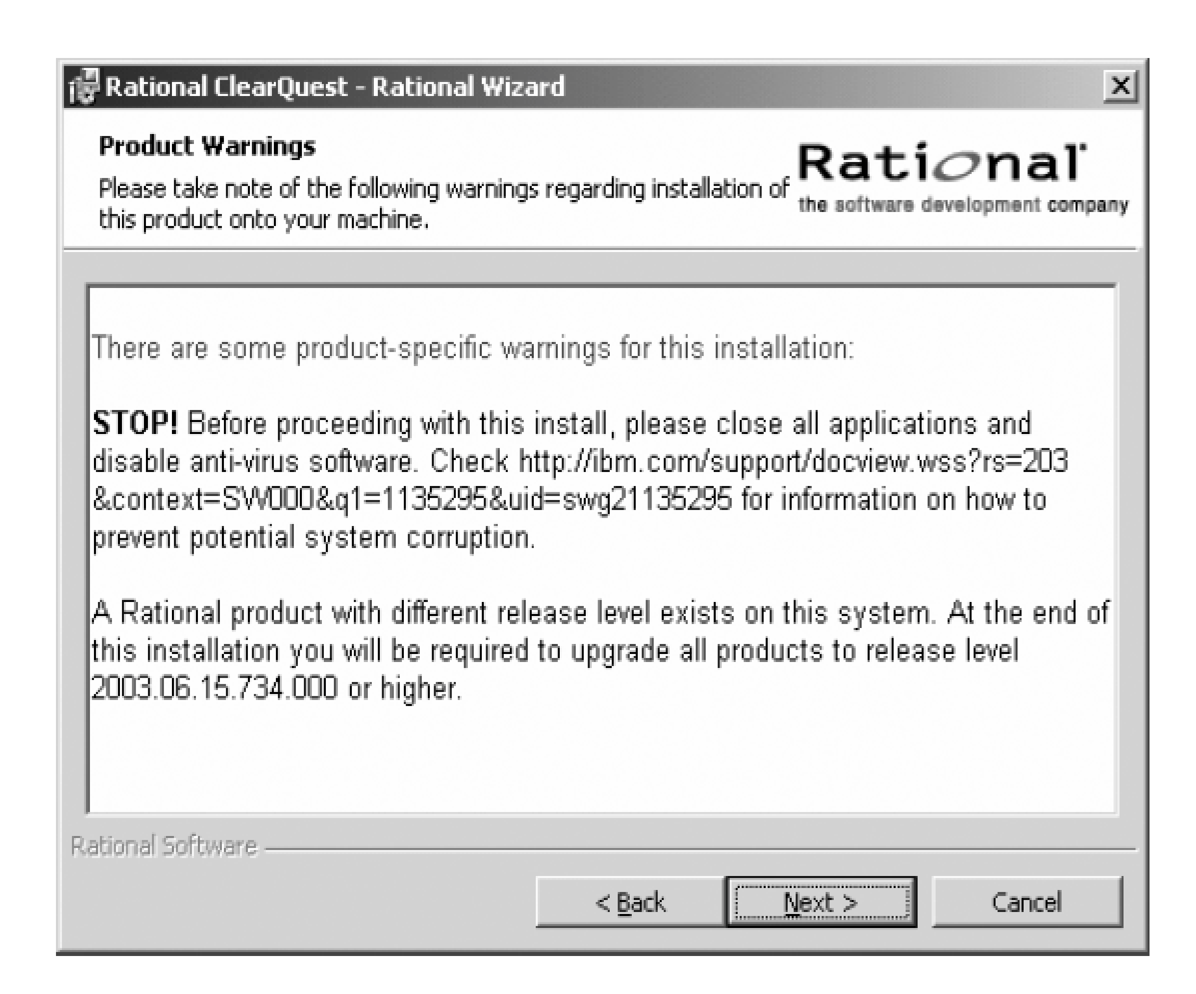


图 10.4　安装图解 4

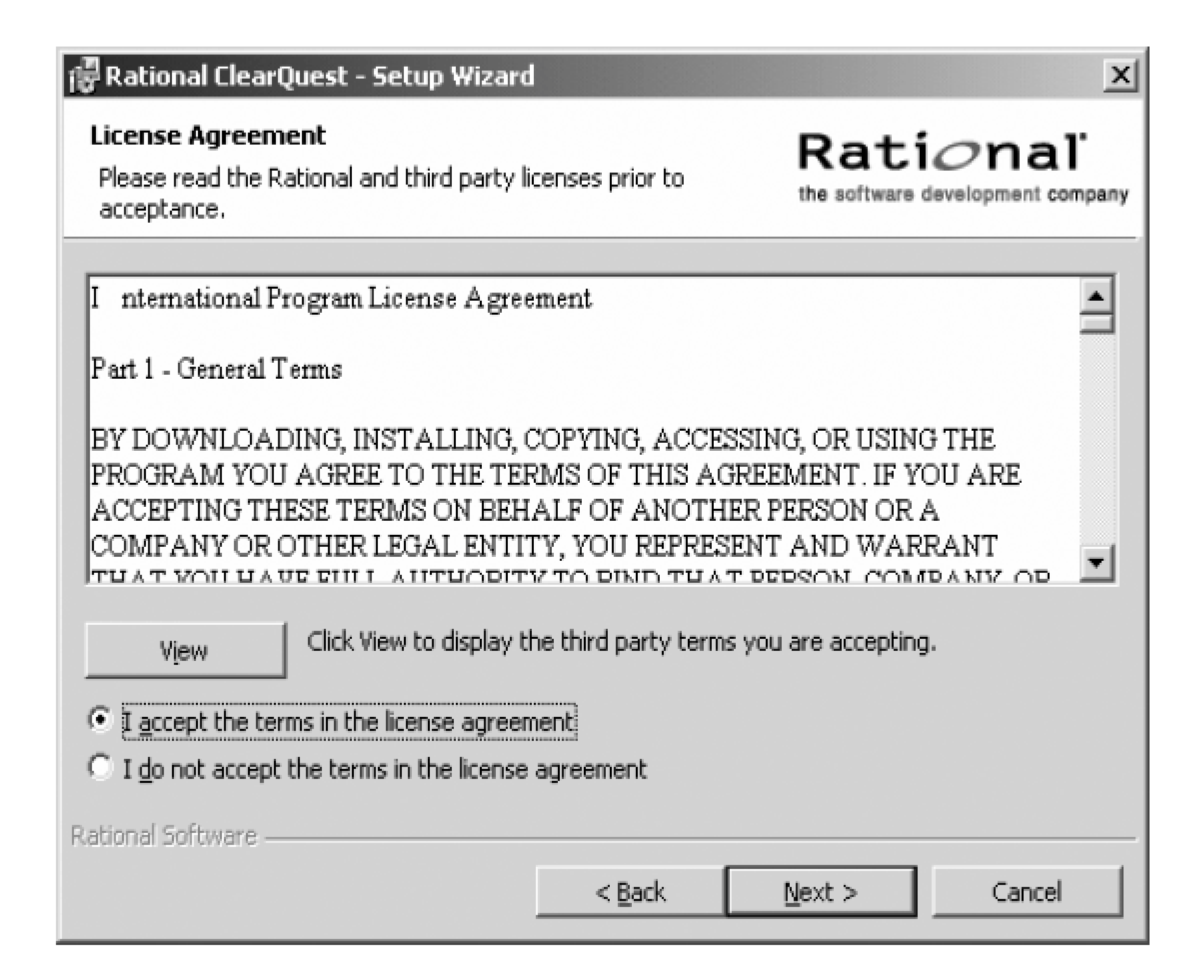


图 10.5　安装图解 5

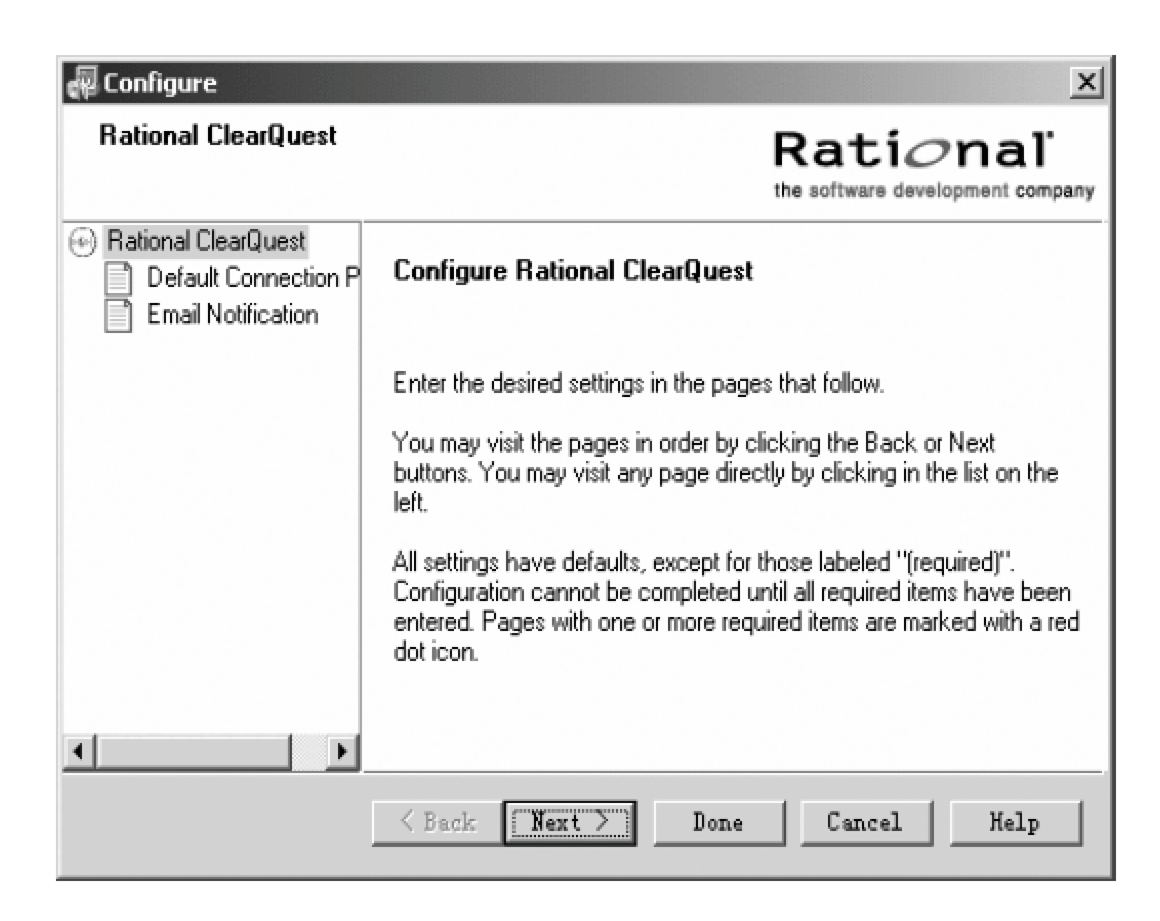


图 10.6　安装图解 6

Default Connection Profile 页面需要对两项内容进行设置，分别是默认数据库和 Email 的设置，如图 10.7 所示。

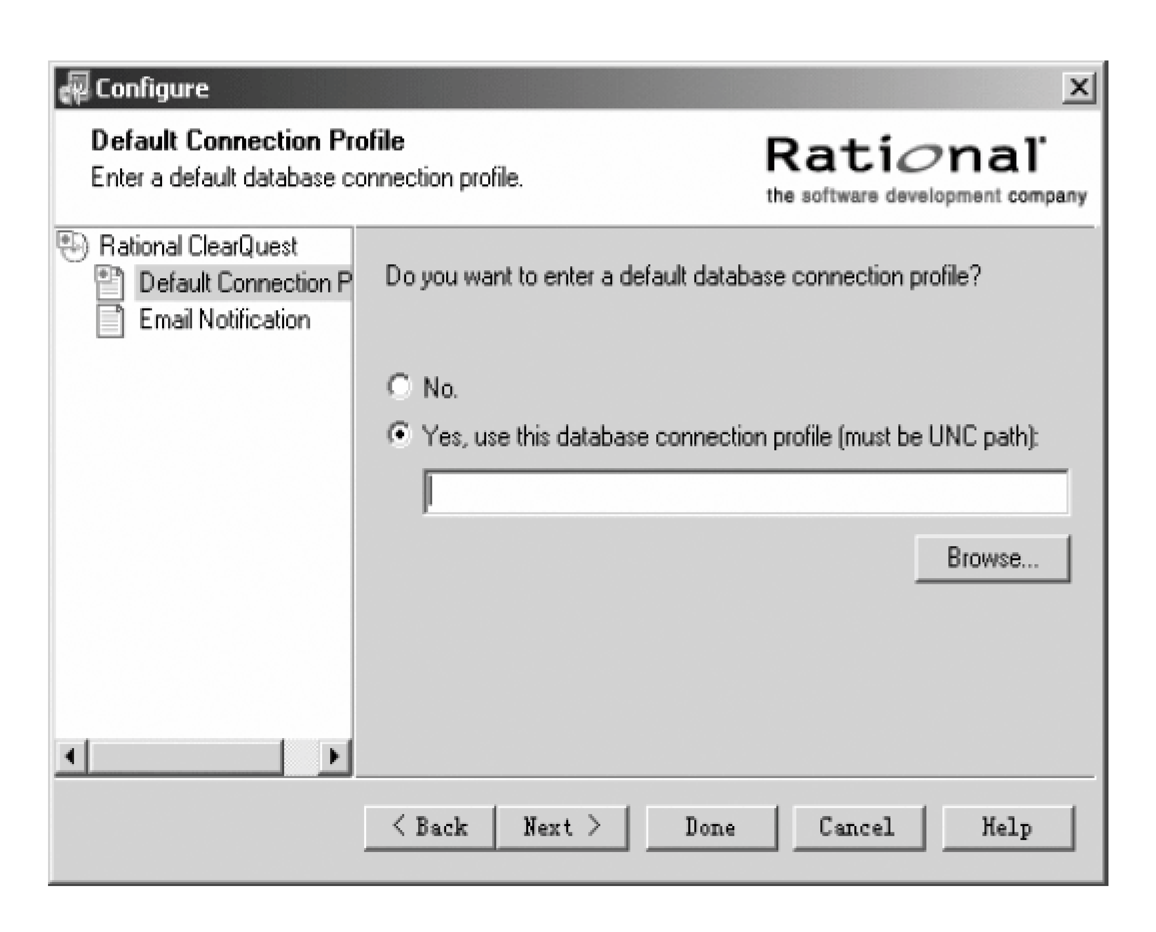


图 10.7　默认数据库连接界面

左侧树形列表中的 Default Connection Profile 项是确认是否设置默认数据库链接：通常是选 No，待安装结束后，到 ClearQuest Maintenance Tool 中新建或导入连接，除非现在知道已有的连接信息。

默认 Email 设置界面如图 10.8 所示。

Email Notification(邮件发布)不需要设置，直接单击 Done 按钮，开始安装 ClearQuest，如图 10.9 所示。

如果想配置 ClearQuest 的 Web 服务器，在安装时，必须选择 Custom(自定义)安装，在 Choose Features 中选中 Web Server Components。

在安装 Rational ClearQuest 的时候，实际就是安装了 ClearQuest 的服务器端和客户端。ClearQuest 的服务器端安装软件与客户端安装软件没有区别。

通常情况下，在服务器端，创建(Create)模式库(Schema Repository)的计算机可以称为服务器端。服务器端创建模式库时，数据库可以选择 SQL Server、Oracle、Access 2000 等。数据库不一定非要安装在服务器端，也可以使用专门的数据库服务器。

客户端也需要安装 Rational ClearQuest。客户端是指连接服务器端创建的模式库，使

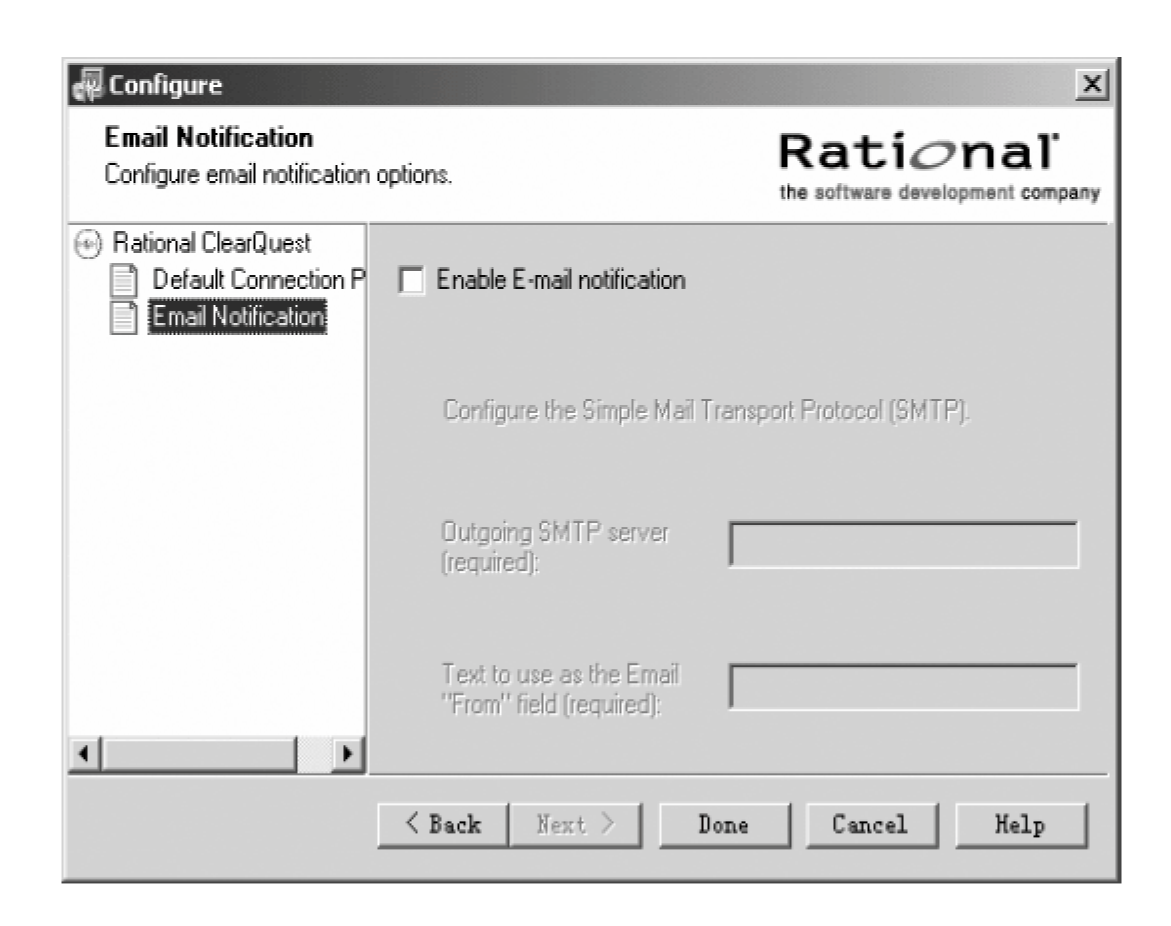


图 10.8　默认 Email 设置界面

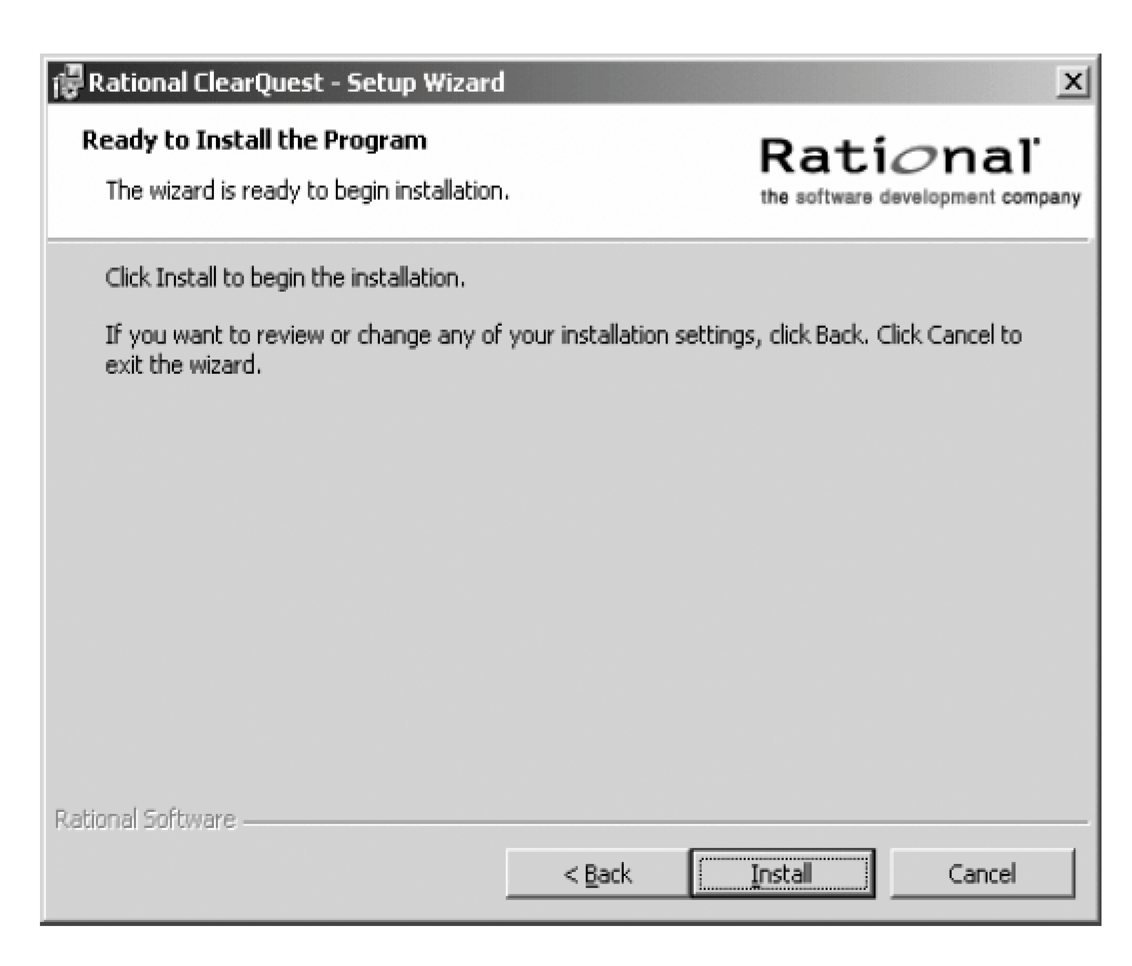


图 10.9　安装图解 7

用 ClearQuest 工具提交缺陷的计算机。同样，如果设置了 Web Server Components，不需要安装，利用 Web 端也可以提交缺陷(访问地址为：http://服务器 IP/cqweb/login)。

10.2　IBM Rational ClearQuest Designer 使用

Rational ClearQuest 功能十分强大，可以和 Rational 系列的其他产品结合，比如 Rational ClearCase、Rational Rose 等。ClearQuest 可以用于变更管理和缺陷跟踪。ClearQuest 由客户端、应用逻辑层以及关系数据库管理系统等部分组成，其中客户端支持 Windows、UNIX 以及 Web 和 Email 等不同的形式。其架构图如图 10.10 所示。

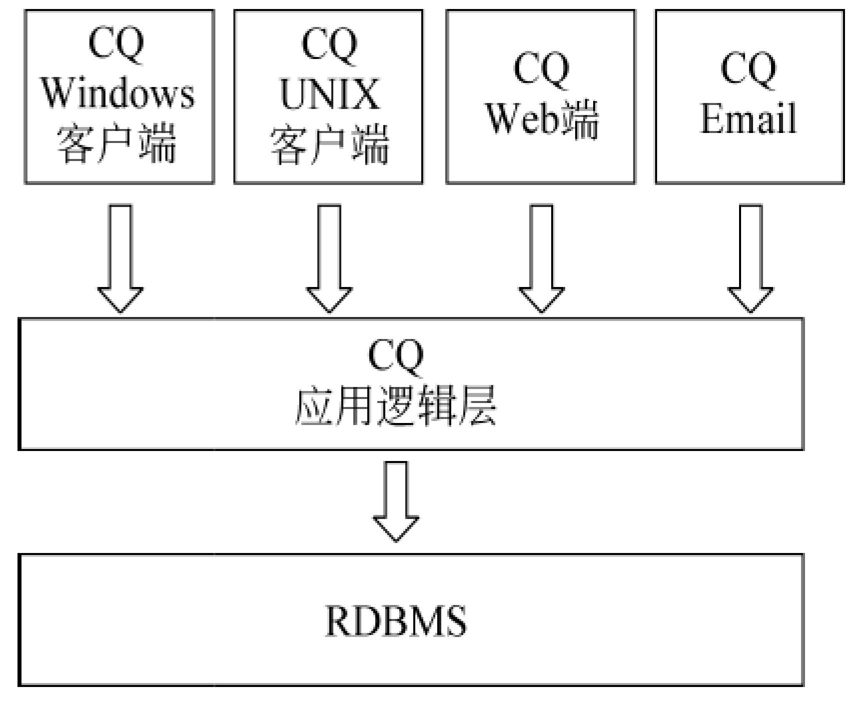


图 10.10　ClearQuest 架构

通过 ClearQuest 架构图可以看出，ClearQuest 可以支持 Web 方式、客户端方式以及 Email。其中客户端方式需要按照 10.1 节描述的操作步骤在每一个需要的客户端进行安装。假如客户端需要使

用 Web 方式,那么客户端必须配备一个支持 ClearQuest 的 Web 浏览器,比如 IE5 版本以上,并在主机配置 IIS 服务。当然客户端也可以选择 Email 方式,那么作为 ClearQuest Administration(管理员)需要配置 Email 环境。

ClearQuest 的应用逻辑层主要包括一些 API(Application Program Interfaces)用于支持 ClearQuest 数据库。

RDBMS(Relational DataBase Management System,关系数据管理系统)主要存储了变更需求以及元数据。

接下来介绍 ClearQuest 的基本操作,这里从 3 个方面来分析。

首先从服务器端开始,服务器端需要创建模式(schema),这里模式主要是定义了软件缺陷和缺陷状态等基本信息。除此之外,服务器端在定制好的模式下需要创建数据库。数据库里面会存放客户端提交上来的不同的缺陷,也就是元数据,以及变更信息等。这一部分会在 10.2.1 节中介绍。

服务器端是使用 ClearQuest 之前的第一步操作,必须对缺陷进行相应的定义并创建数据库后才能在客户端进行缺陷提交、分析等操作。

服务器端设置完毕,接下来需要定义 ClearQuest 用户,ClearQuest 对用户权限等功能设计得比较完整。它主要是为不同客户端的用户定义不同的权限,当然也可以完成添加用户、添加组和调整组等操作。这一部分会在 10.2.2 节中介绍。

设置好用户以及服务器端后,接下来就可以使用 ClearQuest 的客户端进行缺陷管理操作,当然这也是作为 ClearQuest 用户常用的模块。这一部分会在 10.3 节中介绍。

10.2.1 创建模式(Schema)

在应用 ClearQuest 前,必须创建模式库。模式即软件不同缺陷属性及状态等内容,模式库由操作系统已安装的数据库软件管理。创建模式库,需要应用数据库软件创建好空的数据库,并分配拥有权限的数据库登录用户,然后通过维护工具进行创建,并进行数据库的连接。这一部分是利用 ClearQuest Designer(维护工具)来完成的。

启动 ClearQuest Designer,在 ClearQuest 的登录对话框中输入用户名和密码。

(1) User Name 输入 admin。

(2) Password 为空(如果没有改变的话,这个用户是默认的管理员)。

1. 创建新的模式

利用 ClearQuest Designer,可以以一个已经存在的模式为模板创建一个新的模式。所有的模式都被保存在模式库中。接下来利用下述操作流程,创建一个以 TestStudio 模式为模板的新的模式。如果 TestStudio 模式不能使用,可以选择 Common 模式来代替。

(1) 选择菜单"文件"→"新建模式",打开"新建模式"对话框。

(2) 从模式列表中选择 TestStudio,版本为 1,单击"下一步"按钮。

(3) 在"模式名称"栏中输入模式的名称。

(4) 在"注释"文本框中输入 Tutorial schema,单击"完成"按钮。

(5) 出现"是否创建一个与此模式关联的数据库"时,单击"否"按钮(在下一个步骤中将创建数据库)。

(6) 出现"是否检出模式用于编辑",单击"否"。后面在设计模式时需要检出模式。也

可以单击"是",那么接下来就可以在 TestStudio 模式的基础上定制模式。如图 10.11 至图 10.13 所示。



图 10.11　新建模式——步骤 1



图 10.12　新建模式——步骤 2



图 10.13　新建模式——完成

2. 检出已创建的模式

1) 检出一个模式

ClearQuest 在模式库中存储着已有模式的所有版本,必须从模式库中检出模式的最新版本。此后,用户的所有操作都将使用这个被检出的模式。在 ClearQuest Designer 中,选择"文件"→"打开模式"命令,在"打开模式"对话框中选择检出模式。

选择新创建的模式,单击"下一步"按钮。在备注中输入 Adding defect record field to the record form and a new state and action;单击"完成"按钮。

ClearQuest Designer 窗口左侧显示模式的工作区间。请注意所工作的模式的版本号为版本 2。当打开一个模式进行编辑时,ClearQuest Designer 为这个模式自动创建一个新版本,如图 10.14 所示。



图 10.14　检出新定义的模式

2）修改模式

(1) 字段管理

创建新的字段并设置其属性。如图 10.15 所示。



图 10.15　修改字段

在工作区，展开“记录类型”→“Defect”并且双击“字段”，显示“字段”网格。

“字段”网格显示了记录中的所有字段的属性，包括字段名称、类型、缺省值和许可权等。选择“编辑”→“添加字段”命令，打开字段属性对话框，如图 10.16 所示，选择“常规”标签，在字段名称中输入名称。在“类型”中选择 INT，因为该字段包含整数，默认的数据类型是 SHORT_STRING。



图 10.16　修改 defect 字段

也可以直接在“记录类型”上右击，选择“添加”命

令建立一个新的记录类型。

注意：这时类型为 INT 的新创建的字段显示在字段网格中的底部。

(2) 为字段增加一个挂钩(hook)

挂钩是在指定时间(触发器)自动执行的代码部分的进入点，它扩展了 ClearQuest 的功能。为上一步骤中生成的字段添加一个检验有效性的挂钩，这个挂钩用来验证新建字段的值。展开"记录类型"项，双击"字段"。在字段网格中，单击新建字段中的 Validation 单元格，然后单击下拉箭头，选择 SCRIPTS/BASIC，打开脚本编辑器，如图 10.17 所示。脚本编辑器以灰色输出行的形式提供一个 Visual Basic 脚本框架。在脚本编辑器的顶部，"字段"选项显示字段名称，并且"挂钩类型"选项显示 FIELD_VALIDATION。如果显示的不是这些，通过滚动选中这些选项。输入以下代码到脚本编辑器中注释行的下面：

```
REM End If
Dim value_info
Set value_info=GetFieldValue(fieldname)
If Not IsNumeric(value_info.GetValue) Then
User_number_Validation="Must be an integer between 1 and 100"
ElseIf (value_info.Getvalue<1) or (value_info.Getvalue>100) Then
User_number_Validation="Must be between 1 and 100"
End If
```

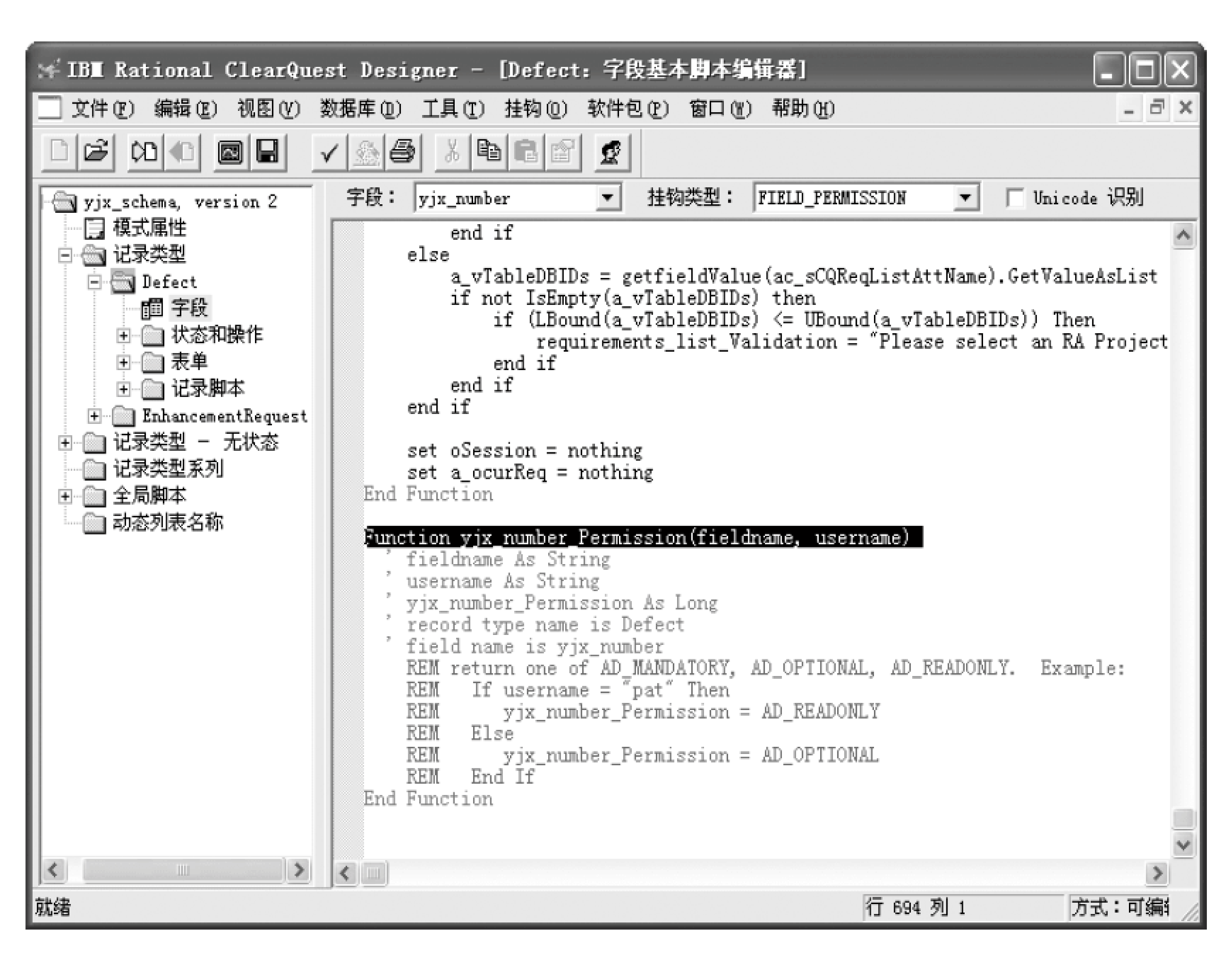


图 10.17　脚本编辑器

以上代码的作用是：当用户在相应字段中输入一个数字时，ClearQuest 客户端运行相应的验证挂钩。如果验证挂钩返回一个非空字符串，用户将被提示该字段中包含无效值。

编译脚本并检测确定没有语法错误。如果没有错误，在 Script Errors 框中将会显示 No error(s) found。

(3) 表单界面的设计

ClearQuest 通过使用窗体来联系一个记录类型并显示相关信息。ClearQuest 也允许用户使用窗体来提交新的相应的记录。刚才创建的字段对用户而言是不可见的，因为它没有出现在任何窗体中。在本步骤中，将在提交窗体中添加这个新建字段，从而使用户可以在提交缺陷报告时确定他们的用户编号。在 ClearQuest Designer 窗口工作区内，展开记录类型→Defect→表单并双击 Defect_Base_Submit，打开 Defect_Base_Submit 窗体的同时，字段列表和 Controls Palette 同时也被打开，如图 10.18 所示。

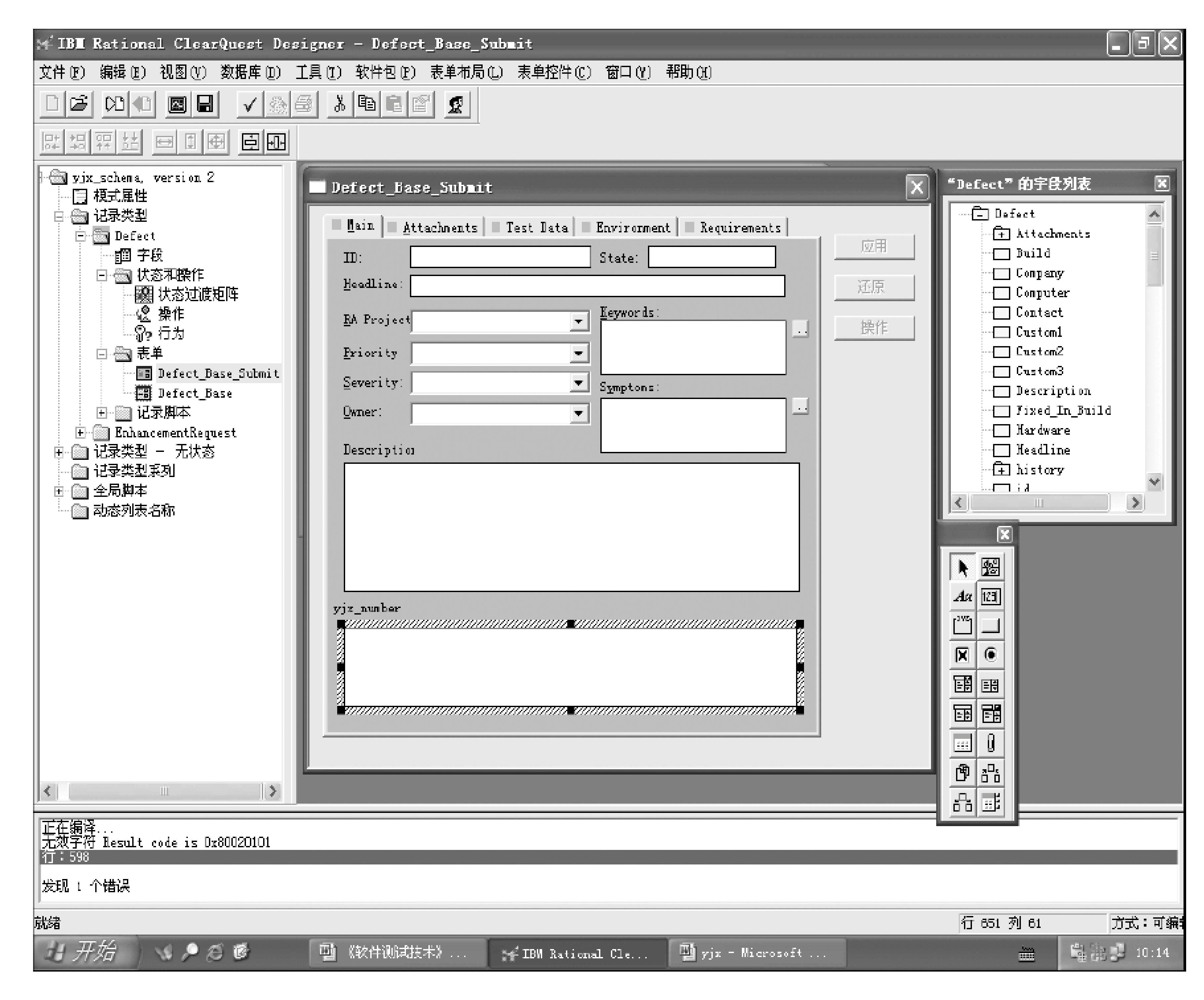


图 10.18 修改窗体

如果需要的话，最大化 ClearQuest Designer 窗口，拉伸 Defect_Base_Submit 窗口以便在窗口的底部可以添加一个新的字段。在字段列表内，滚动并找到新建的字段。单击字段名称(而不是图标)并拖动至 Defect_Base_Submit 窗体的底部。字段名称和一个文本框出现在窗体中。ClearQuest 为字段自动选择一个文本框控件。双击文本框显示属性页，如图 10.19 所示，浏览控件提供的标签和控制功能，然后单击"确定"按钮关闭属性页。工作区内，右击 Defect_Base_Submit。确定 Submit Form 项在快捷菜单中已被选中。如果该项目没有被选中，单击并添加该选项。关闭 Defect_Base_Submit 窗口。



图 10.19 修改窗体参数

ClearQuest 使用窗体与记录类型进行联系，并使用该窗体显示相应记录类型的详细信息。每一个记录类型可以包含一个或多个窗体：记录窗体和提交窗体。提交窗体不是必需的，但是每一个模式必须包含一个记录窗体。

(4) 缺陷状态设定

ClearQuest 提供了状态过渡矩阵功能帮助用户完成状态的设计工作。在“状态过渡矩阵”上右击，选择“打开”命令，出现如图 10.20 所示的窗口。

To \ From	Submitted	Assigned	Opened	Resolved	Closed	Duplicate	Postponed
Submitted							
Assigned	Assign						Assign
Opened		Open		Reject	Re_open		
Resolved			Resolve				
Closed	Close	Close		Validate			Close
Duplicate	Duplicate		Duplicate	Duplicate	Duplicate		
Postponed	Postpone	Postpone	Postpone				

图 10.20 缺陷状态管理

在状态过渡矩阵中，ClearQuest 以二维的矩阵表示状态的变化，From 表示源状态，以列的方式显示；To 表示目标状态，以行的方式显示。中间白色区域表示连接状态的动作或操作。有了状态，需要设定动作将两个或多个状态连接起来，在 ClearQuest 中由“操作”完成此功能。在“操作”上右击，在弹出的快捷菜单中选择“打开”命令，可添加新的操作。

可以按照以下步骤添加一个新的状态以及变化到新状态的新的操作。

① 添加新的状态 Reassigned。

选择“编辑”→“添加状态”，显示“添加状态”对话框，如图 10.21 所示。在“名称”栏输入 Reassigned，单击 OK 按钮。Reassigned 状态在状态转换矩阵中的行和列同时出现。

② 添加新的操作 Reassign。

选择“编辑”→“添加操作”。在“Defect 操作”对话框中，选择“常规”标签，在“名称”栏内输入 Reassign。ClearQuest 将操作类型默认设置为 CHANGE_STATE，如图 10.22 所示。关闭“Defect 操作”对话框。Reassign 操作在操作网格的底部出现。



图 10.21 添加状态



图 10.22 添加操作

前面的步骤中，所创建的 Reassign 操作类型是 CHANGE_STATE。只有类型为 SUBMIT、CHANGE_STATE 或 DUPLICATE 的操作才可以开始状态转换。在这个步骤

中，将为 Reassign 操作通过指定源状态及目的状态来定义一个状态转换。其结果是：ClearQuest 客户端用户能够通过选择操作 Reassign 将所有记录为 Opened 和 Resolved 的状态转换为 Reassigned 状态。操作步骤如下。

展开“记录类型”→Defect→“状态和操作”，双击“操作”。右击行标签 Reassign，在快捷菜单中选择“操作属性”，出现“Defect 操作”对话框。在“状态”标签内，选择 Opened 和 Resolved 作为源状态。选择 Reassigned 作为目的状态，如图 10.23 所示。

关闭“Defect 操作”对话框，应用该状态转换。双击工作区内的状态过渡矩阵，检查 ClearQuest 是否应用了 Reassigned 状态的转换，如图 10.24 所示。



图 10.23　定义转换

To \ From	Submitted	Assigned	Opened	Resolved
Submitted				
Assigned	Assign			
Opened		Open		Reject
Resolved			Resolve	
Closed	Close	Close		Validate
Duplicate	Duplicate		Duplicate	Duplicate
Postponed	Postpone	Postpone	Postpone	
Reassigned			Reassign	Reassign

图 10.24　查看状态转换

状态转换矩阵显示，无论原来是 Opened 状态还是 Resolved 状态，Reassign 操作都将转换记录为 Reassigned 状态。

10.2.2　设计数据库

在模式库下建立产品库和测试库。产品库用来存放项目，可以一个产品对应一个项目，也可以多个项目同在一个产品库中。测试库用来对 ClearQuest 进行定制式的测试，一般测试过程时不需要。

接下来创建一个新的数据库并且将其关联至新的模式上。

数据库是 ClearQuest 客户端使用的变更请求记录数据库。选择菜单“数据库”→“新建数据库”命令，打开“新建数据库”对话框。在“逻辑数据库名称”中输入 my_db。逻辑数据库名字的字符长度在 1～5 之间。单击“下一步”按钮。供应商选择 MS_ACCESS。在物理数据库名称中输入 c:\my_db 作为数据库的完整路径名称(路径中指定的目录必须存在。例如，如果指定了 c:\temp\my_db，那么 temp 目录必须存在。如果数据库不存在，则 ClearQuest 会在指定的目录中创建)。选择“生产数据库”选项，单击“下一步”按钮。在模式列表中选择新建的模式，单击“完成”按钮。出现数据库创建成功提示消息窗口时，单击 OK 按钮，如图 10.25 所示。

ClearQuest Designer 创建 my_db 数据库并且使用新建的模式进行初始化。

下面介绍创建测试数据库的步骤。

选择菜单“数据库”→“新建数据库”命令，打开“新建数据库”对话框。在“逻辑数据库名称”中输入 my_db，单击“下一步”按钮。“供应商”中选择 MS_ACCESS。在“物理数据库名称”中输入 c:\my_db 作为数据库的完整路径名称。选择“测试数据库”选项，单击“下一步”按钮。在模式列表中选择新建的模式，单击“完成”按钮。在出现数据库创建成功提示消息



图 10.25　新建数据库

窗口时，单击 OK 按钮。

上述过程如图 10.26 至图 10.28 所示。



图 10.26　创建测试数据库——定义数据库名



图 10.27　创建测试数据库——定义数据库类型和路径



图 10.28 创建测试数据库——设置参数

10.2.3 用户及权限管理

ClearQuest 的用户权限控制分为两部分，一部分是对单个用户设定特权，定义用户是否具有对产品进行定制设计、查询设计和建立文件夹等权限。另一部分是对用户组设定操作权限，定义用户组是否具有建立项目、建立模块、建立邮件规则以及对缺陷状态进行改变等权限。对单个用户的权限设定在用户管理部分完成；对用户组的权限设定，除在用户管理部分设定相关功能外，还需要在设计器中进行权限设置。

用户管理与模式是相互独立的，因此不用检出一个模式来添加新用户。

1. 添加用户

为系统增加新用户并定义这个用户可以访问的数据库。在 ClearQuest Designer 中，选择“工具”→“用户管理”命令，打开“用户管理”对话框，如图 10.29 所示。单击“用户操作”按钮并单击“添加用户”，打开“添加用户”对话框。当详细说明用户信息时，可以输入用户描述



图 10.29 用户管理主界面

及电话号码。如果系统支持电子邮件通知，ClearQuest 使用用户描述信息为设定的注册 ID 号确定正确的邮件地址，如图 10.30 所示。



图 10.30 “添加用户”对话框

2. 设置用户权限

可以在任何时候编辑用户属性，可以变更用户名称、登录名、口令、电话、电子邮件、描述和访问权限等，也可以改变用户订阅的数据库。用户对其订阅的数据库有访问和修改的权限。

在 ClearQuest Designer 中，选择“工具”→“用户管理”，也可以编辑用户的信息。

3. 升级数据库

在 ClearQuest 设计器中对权限进行设定后，需要检出设定模式，才能令用户具有设定好的权限。这也就是所谓的数据库升级操作。

在数据库菜单中进行升级数据库的操作，完成测试库数据到相应产品库的同步更新。

在用户管理界面进行设置后，要选择“数据库操作”中的“升级”进行相应的所属数据库数据更新，如图 10.31 所示。



图 10.31 升级数据库

注意：ClearQuest 的用户管理不存在删除功能，所以增加用户及用户组时请慎重。

10.3 IBM Rational ClearQuest 客户端使用

10.3.1 缺陷变更管理

这里使用 ClearQuest 的客户端来进行缺陷的提交和管理。图 10.32 是 ClearQuest 客户端缺陷的管理流程，也就是常用的缺陷的状态变换过程，可供读者参考。

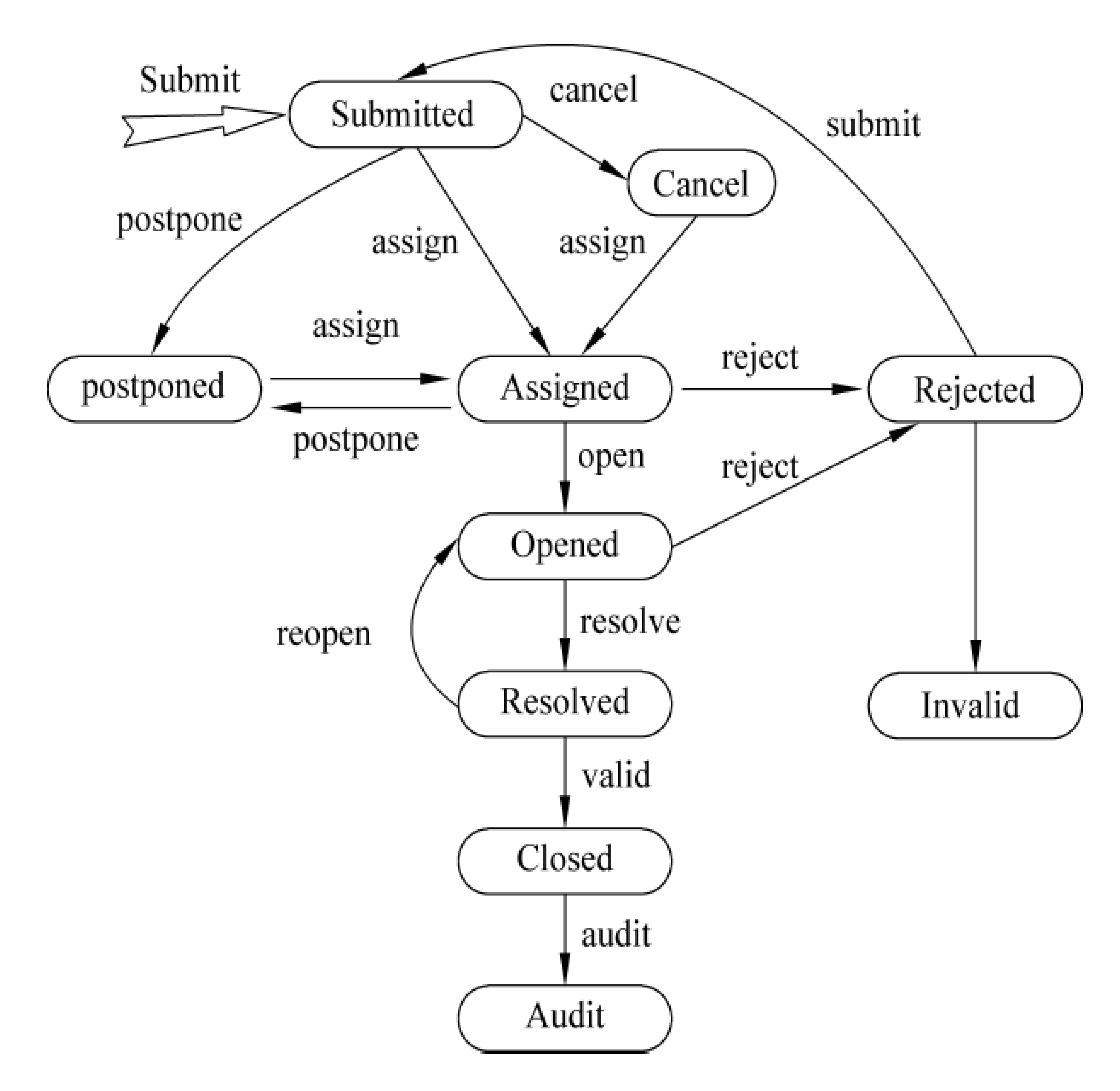


图 10.32　ClearQuest 中默认的缺陷状态变更

首先,需要打开 IBM Rational ClearQuest 客户端,输入用户名、密码以及有权限访问的数据库名称,进入 IBM Rational ClearQuest 客户端主界面,如图 10.33 所示。

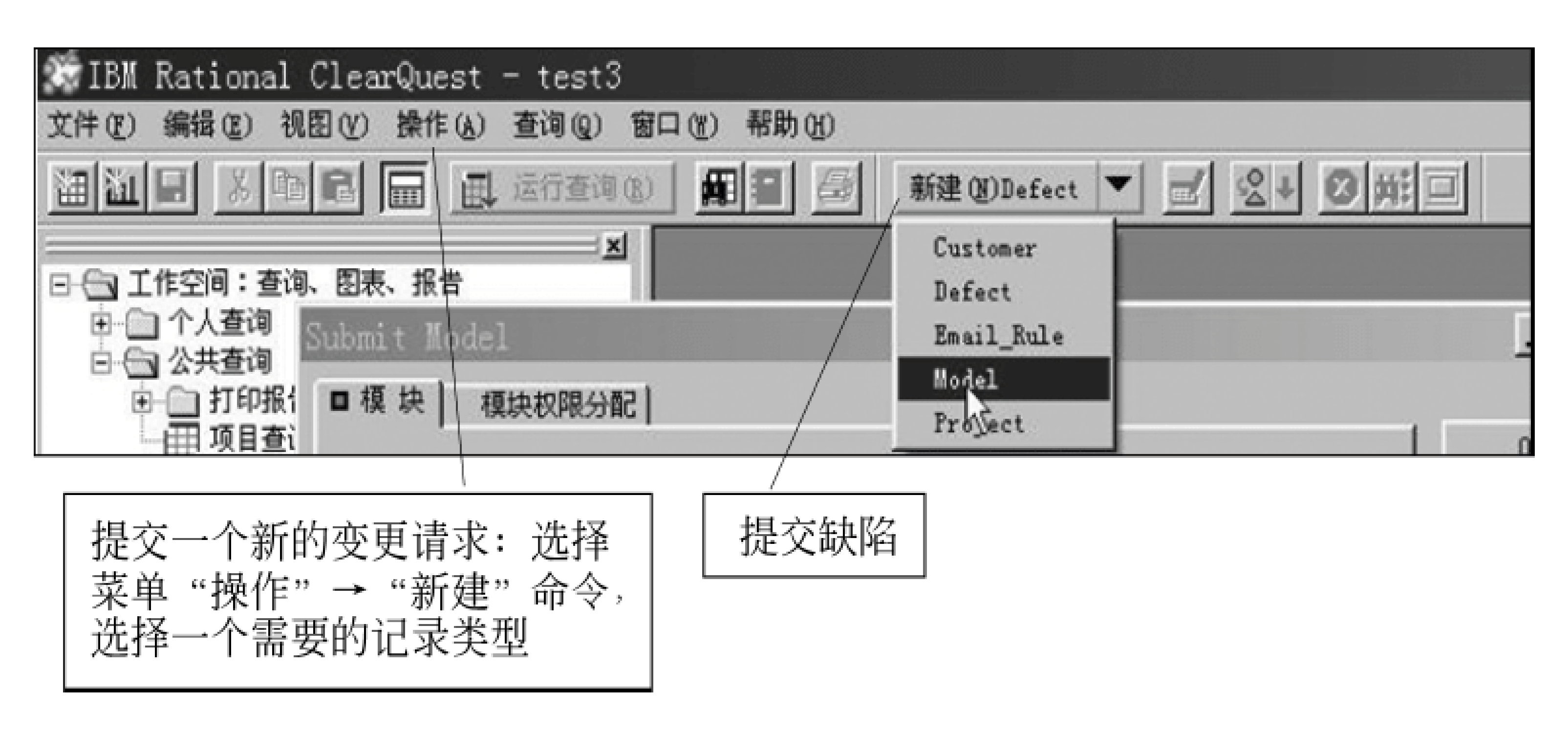


图 10.33　ClearQuest 客户端主界面

如果要提交一个缺陷,有 3 种方法可提供。

(1) 选择菜单"操作"→"新建"命令,选择一个记录类型进行提交。

(2) 单击"新建 Defect",并选择一种记录类型,完成提交。

(3) 单击"新建 Defect"旁边的箭头,选择一种记录类型,完成提交,如图 10.34 所示。

10.3.2　创建公共查询和图表

拥有创建查询权限的用户可以创建公共查询,每个用户都可以创建个人查询。首先右击"公共查询",在快捷菜单中选择"新建查询"命令,如图 10.35 所示。选择要查询的记录类型,例如 Defect 类型,如图 10.36 所示。

选择好要查询的记录类型后,可以重新定义一次新的查询,也可以在原来的某次查询定义基础上进行查询,如图 10.37 所示。

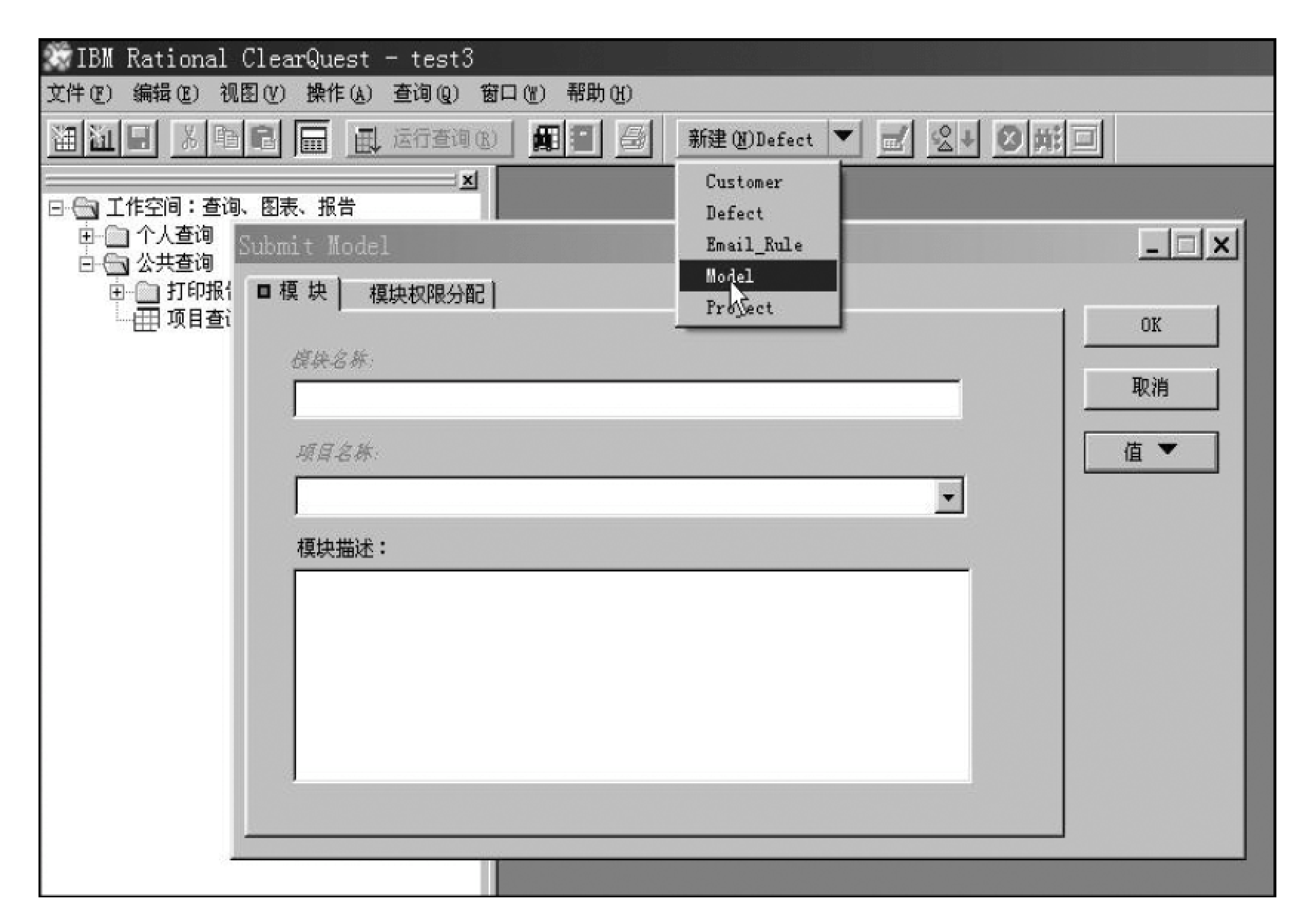


图 10.34　ClearQuest 客户端提交缺陷



图 10.35　ClearQuest 客户端按用户特殊需求进行查询设置

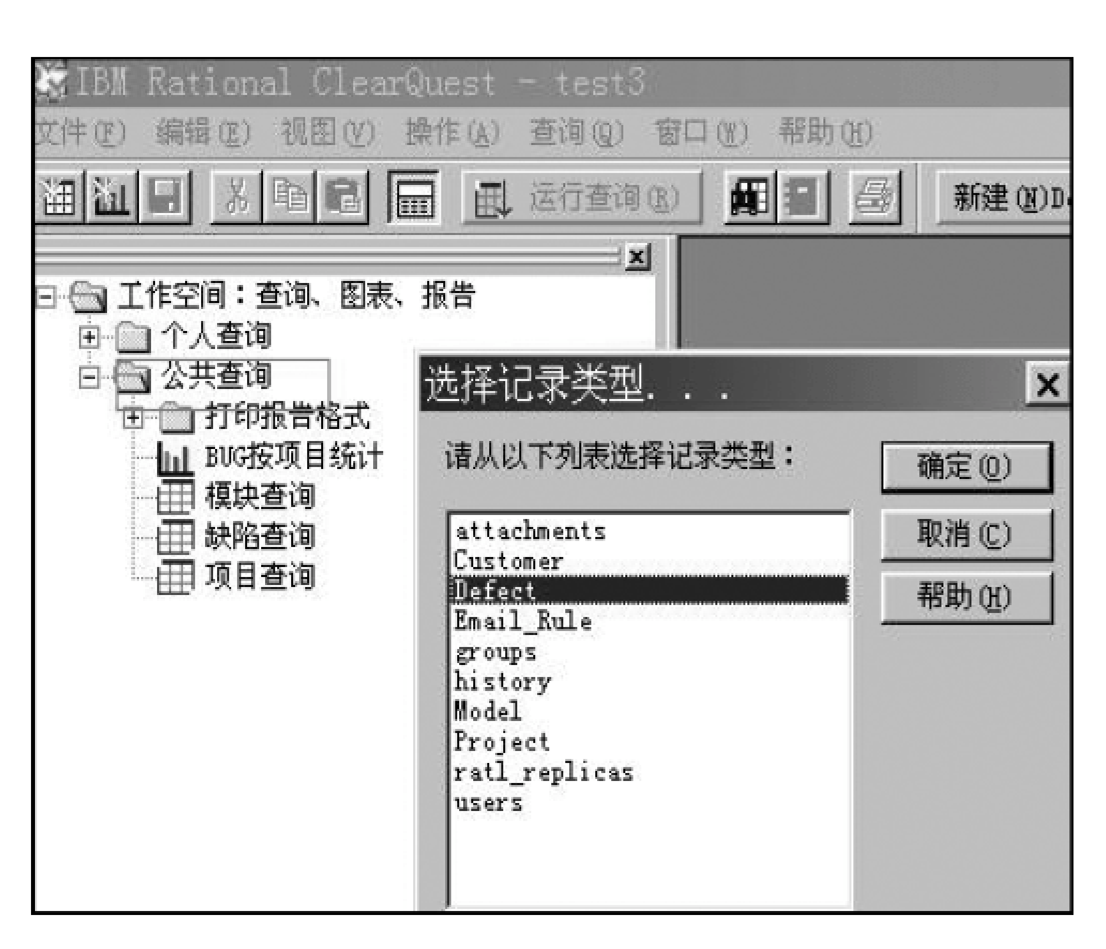


图 10.36　查询设置 1

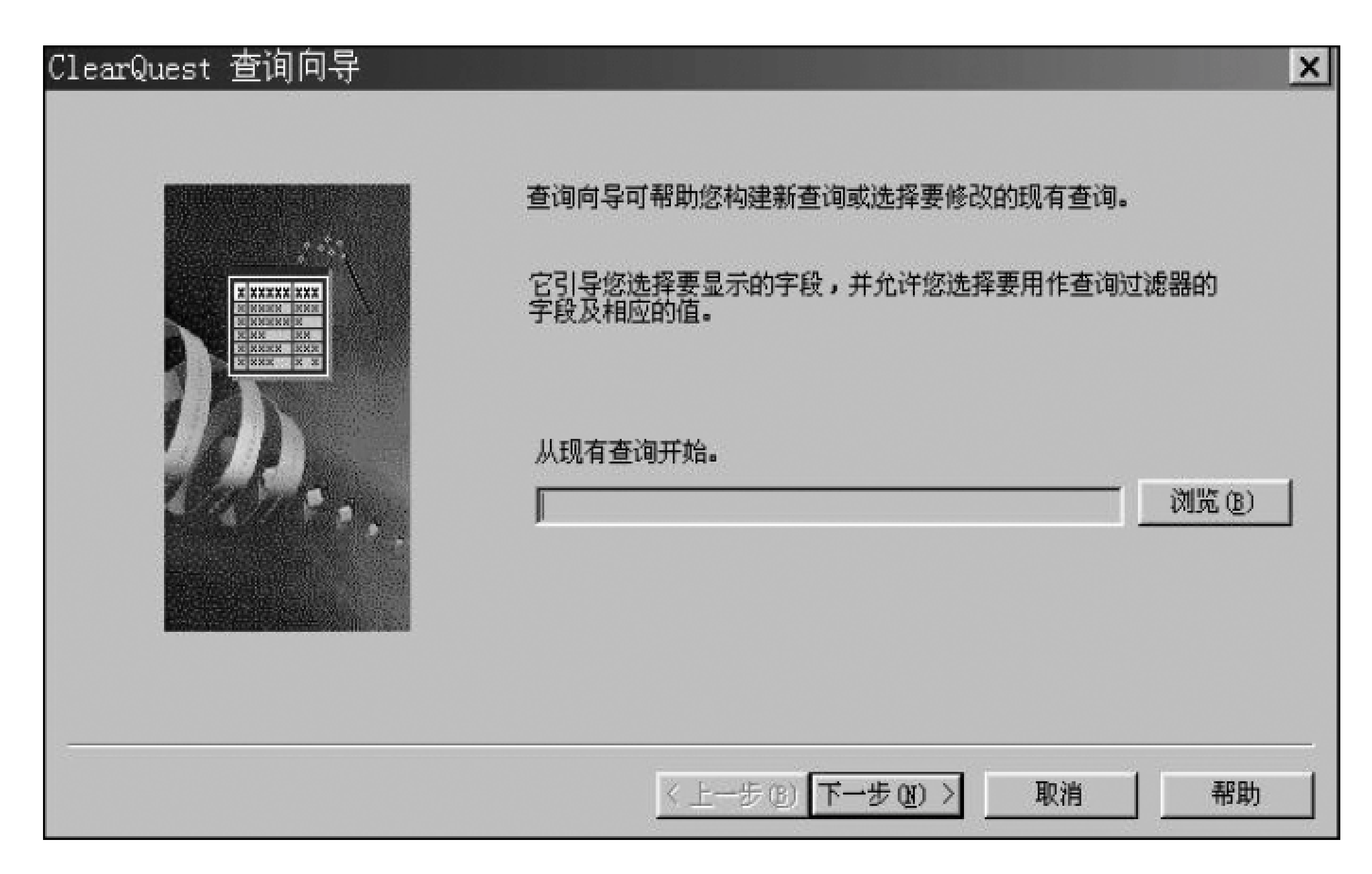


图 10.37　查询设置 2

接下来可以定义查询后的显示方式，选中的字段会放到网格中，并在查询中显示，未选中的字段会被隐藏，如图 10.38 所示。

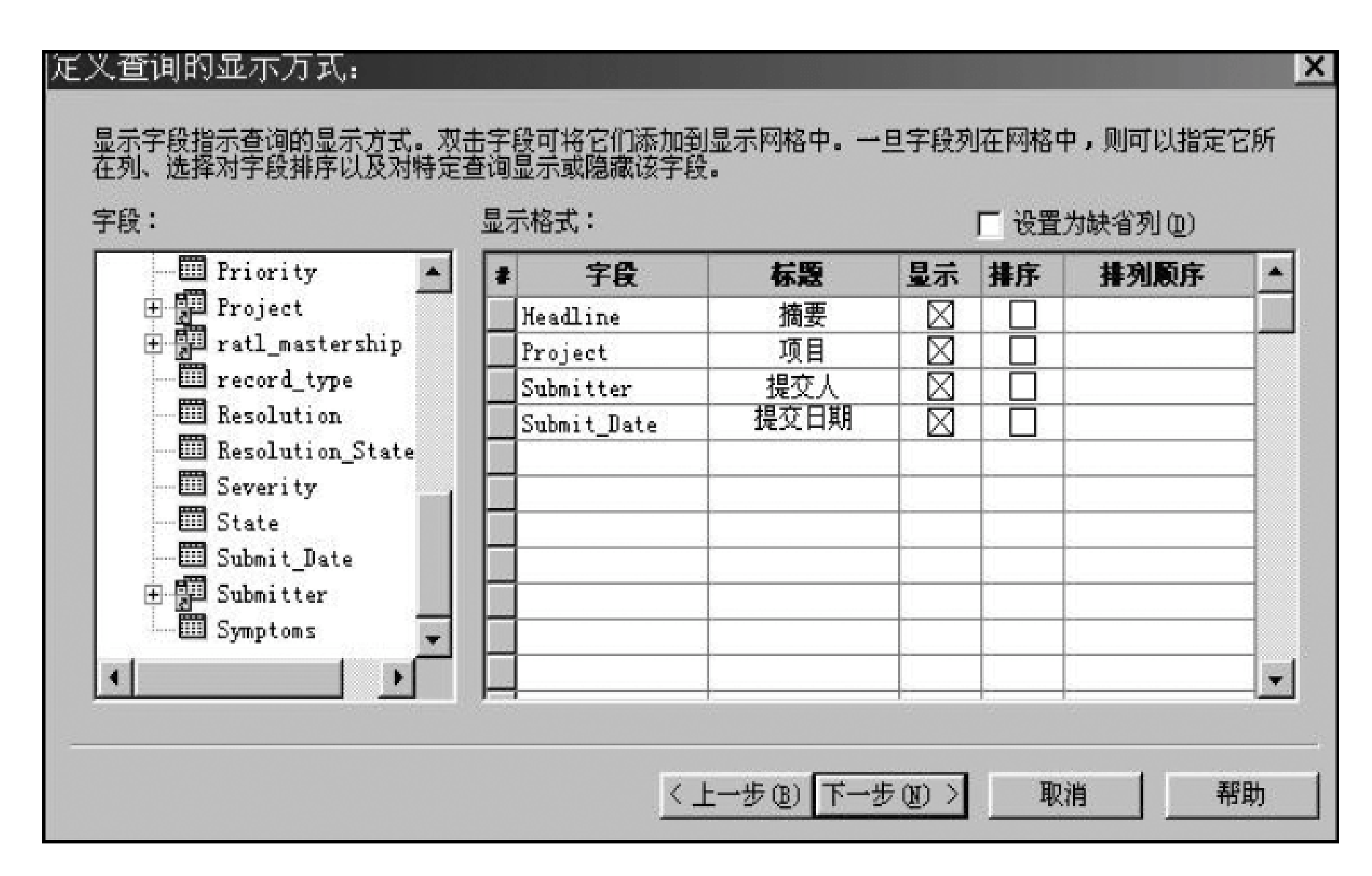


图 10.38　查询设置 3

利用过滤器可以定义查询的条件，如图 10.39 所示。



图 10.39　查询过滤器

设置查询完毕后，可以直接运行设置好的查询，也可以通过主界面的运行查询快捷键完成。

另外，ClearQuest 支持图表的表示。右击“公共查询”，在快捷菜单中选择“新建图表”命令。读者可以自行尝试。

10.4 本章小结

本章介绍了 IBM Rational ClearQuest 软件的安装和基本使用流程。IBM Rational ClearQuest 是一款缺陷变更管理工具，主要是针对繁杂的测试管理而设计开发的。本章首先介绍了 ClearQuest 的安装。之后分析了 ClearQuest 的使用流程。使用 ClearQuest 之前需要 ClearQuest 管理员通过 ClearQuest Designer 来创建模式库和数据库。数据库分为产品库和测试库，产品库用来存放项目，可以一个产品对应一个项目，也可以多个项目同在一个产品库中；测试库用来对 ClearQuest 进行定制时的测试，正常应用时不需要。ClearQuest 客户可以通过客户端提交缺陷并按照设定好的模式来进行管理。

习 题

1. 什么是软件缺陷？简述缺陷的几种状态。
2. ClearQuest 可以提供几种客户端访问方式？
3. ClearQuest 管理员是否可以更改缺陷状态变化过程？如何操作？
4. ClearQuest 可以建立生产库和测试库，两者有什么区别？

参 考 文 献

[1] Paul C. Jorgensen. 软件测试[M]. 3 版. 李海峰,马琳,译. 北京:人民邮电出版社,2010.

[2] 宫云站. 软件测试教程[M]. 北京:机械工业出版社,2008.

[3] Glenford J. Myers,Tom Badgett,Corey Sandler. 软件测试的艺术[M]. 3 版. 张晓明,黄琳,译. 北京:机械工业出版社,2012.

[4] 朱少民. 软件测试方法和技术[M]. 北京:清华大学出版社,2005.

[5] Ron Patton 软件测试[M]. 张小松,王钰,曹跃,译. 北京:机械工业出版社,2006.

[6] 黎连业,王华,李龙,等. 软件测试技术与测试实训教程[M]. 北京:机械工业出版社,2012.

[7] 吴建,郑潮,汪杰. UML 基础与 Rose 建模案例[M]. 北京:人民邮电出版社,2005.

[8] 翟长勇. 基于 Petri 网的 Web 应用软件测试技术研究[D]. 贵阳:贵州大学,2009.

[9] 李文瑞. 基于 Petri 网的软件测试技术研究[D]. 无锡:江南大学,2011.

[10] 孙琳,刘久富,杨振兴. 基于 Petri 网的软件测试用例生成方法[J]. 计算机测量与控制,2010 (9):2019-2022.

[11] 郑艳艳. 基于 Petri 网模型的 GUI 软件测试用例生成研究[D]. 武汉:华中师范大学,2006.

[12] 李留英. UML 测试技术的研究与实现 [D]. 长沙:国防科学技术大学,2000.

[13] 李学升. 基于 UML 模型的软件测试技术研究与实现[D]. 北京:中国石油大学,2008.

[14] 李永亮. 基于 EFSM 模型的软件故障检测与一致性测试生成研究[D]. 长沙:湖南大学,2009.

[15] 侯莹. 基于有限状态机的 J2ME 程序 GUI 测试技术[D]. 大连:大连海事大学,2009.

[16] 包健,魏丽娜,赵建勇. 基于有限状态机的电梯控制系统故障诊断方法[J]. 计算机应用,2012,32(6):1692-1695.

[17] 年晓玲. 基于扩展有限状态机软件测试用例自动生成的研究[D]. 成都:西南交通大学,2005.

[18] 宁宁. 软件可靠性模型及其参数估计[D]. 成都:电子科技大学,2008.

[19] 朱厚英. 软件可靠性计量模型及 ELM 算法研究[D]. 杭州:中国计量学院,2012.

[20] 李春玲. 软件可靠性模型研究[J]. 电子产品可靠性与环境试验,2008,26(2):33-36.

[21] 楼俊钢,江建慧,帅春燕,等. 软件可靠性模型研究进展[J]. 计算机科学,2010,37(9):13-19.

[22] 潘浪涛. 铁路自动售票系统软件可靠性研究[D]. 北京:中国铁道科学研究院,2012.

[23] 张治生,陈怀民,吴成富,等. 小型无人机飞控系统软件可靠性设计与建模研究[J]. 计算机测量与控制,2011,19(6):1489-1492.

[24] 丛珉,陆民燕,白云峰. 软件可靠性预计方法研究及实现[J]. 北京航空航天大学学报,2002,28(1):34-38.

[25] 魏娜娣. 软件性能测试——基于 LoadRunner 应用[M]. 北京:清华大学出版社,2012.

[26] 郑人杰. 软件测试[M]. 北京:人民邮电出版社,2011.